基于模型的设计——MCU篇



刘 杰 翁公羽 周宇博/著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

策划编辑：胡晓柏

封面设计：runsign



基于模型的设计——MCU篇

作者简介



刘 杰 毕业于浙江大学信电系通信工程专业，获工学博士学位，现为硕士生导师，兼职教授。长期从事嵌入式器件的研究与开发，特别是近3～4年，夜以继日地潜心钻研基于模型的设计，这项最近几年才在全球掀起的新技术。致力于宣传、推广基于模型的设计在我国的应用和普及，已经出版了国内第一部基于模型设计的专著《基于模型的设计及其嵌入式实现》。

内容精华

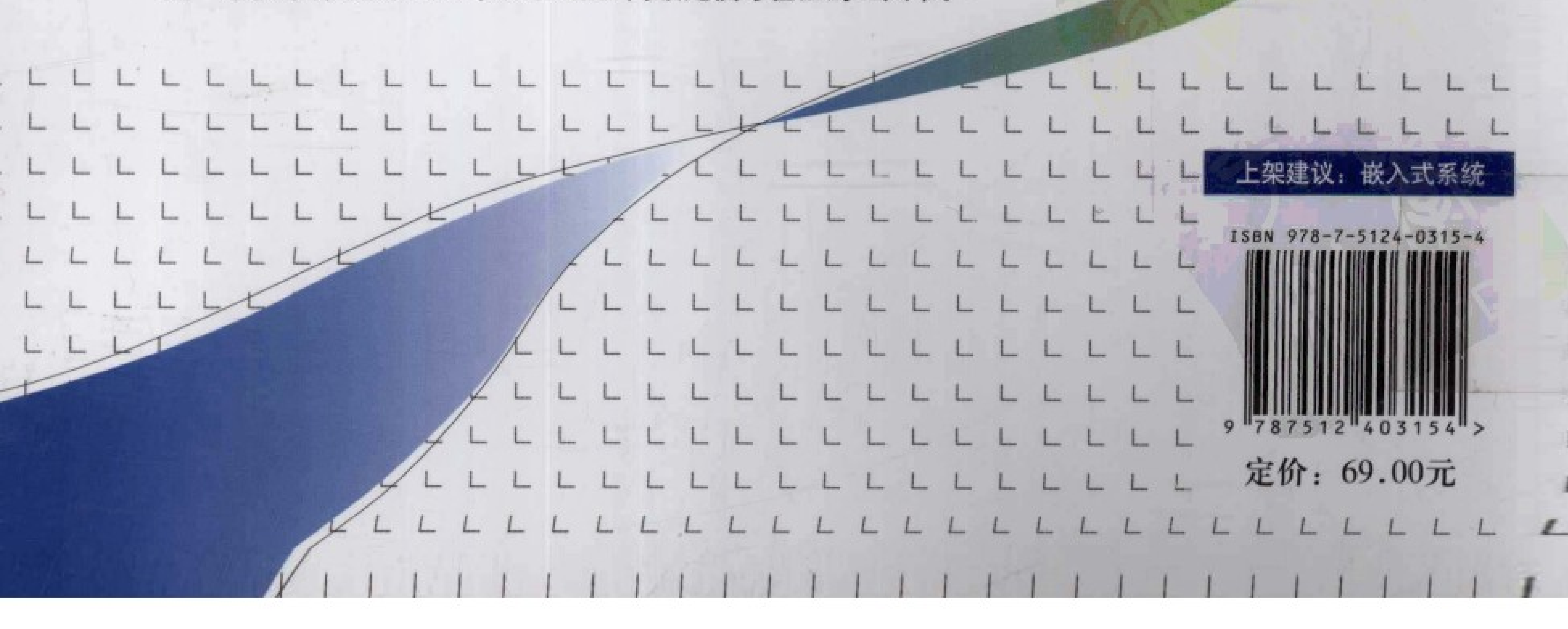
本书采用了先进的产品开发思想——基于模型设计的方法，并以MATLAB R2010b为软件平台。让工程师在可视化的MATLAB统一开发环境中，一边进行需求分析、算法研究、模型与需求分析的双向跟踪、模型验证与优化；另一边进行自动生成C代码的软件在环测试、处理器在环测试、代码的有效性分析、代码与模型的双向跟踪、代码优化、硬件测试等，让算法到嵌入式实时C代码的生成一步到位、一次成功，避免传统开发MCU器件，前期投入大、开发周期长、一般需要重复多次才能成功的弊端。

实现了51单片机、英飞凌C166单片机、dsPIC3x数字信号处理器、ARM处理器的快速开发，其资金投入、工作量和所需花费时间只占传统方法的1/3～1/2，有效地规避MCU应用开发的潜在市场风险。

书中着重介绍了有限状态机Stateflow描述MCU编程的特点，让一些复杂或晦涩的逻辑关系变得异常简单。

读者对象

本书可作为MCU器件开发的技术手册，也可作为高校的嵌入式开发，特别是基于模型设计的教材，也是一本很好的Stateflow和Simulink高级建模与验证的工具书。



上架建议：嵌入式系统

ISBN 978-7-5124-0315-4

9 787512 403154

定价：69.00元

基于模型的设计
——MCU 篇

刘 杰 翁公羽 周宇博 著

北京航空航天大学出版社

内容简介

本书以基于模型的设计在MCU中的应用为主线，分三部分介绍全书。

第一部分，深入剖析了Stateflow的建模与应用，以及Simulink建模与调试；介绍了新版MATLAB的特色功能与R2010b版中Embedded MATLAB的编程规范和新的编程与调试模式；最后着重讨论了用户驱动模块的创建过程与应用实例等。

第二部分，演示了简化的基于模型设计，即基于模型的8051、英飞凌C166、Microchip dsPIC、ARM等MCU中的快速开发，并在Proteus中进行虚拟硬件测试，使读者直观地感受到在可视化的开发环境中，从算法验证到嵌入式C代码自动生成一步到位的方便与高效。

第三部分，以直流电动机的PID控制模型为例，介绍了满足DO－178b航空电子规范的完整基于模型设计在ARM上的实现。其流程包括可执行、可跟踪的需求分析/技术规范、Model Advior测试、系统测试、设计测试、浮点模型到定点模型的自动转换与定标、为特定芯片生成嵌入式C代码、软件/硬件在环测试、嵌入式实时C代码的自动生成，最后是手工底层驱动代码与自动代码的整合等，这部分是本书的总结与核心。

本书可作为航天军工、汽车电子、通信与信息处理，电力等行业的工程师从事MCU开发时的技术手册，也可作为高校电类专业的MCU开发或基于模型设计的教材，同时也是Simulink/Sateflow高级建模与验证的参考书，另外也为广大高校学生（本、硕、博）做毕业设计提供了一种高效、快捷的软件实现方法。

图书在版编目(CIP)数据

基于模型的设计. MCU篇/刘杰等著. -- 北京 ：北京航空航天大学出版社，2011.1

ISBN 978-7-5124-0315-4

Ⅰ. ①基… Ⅱ. ①刘… Ⅲ. ①单片微型计算机—微控制器—程序设计 Ⅳ. ①TP332.3

中国版本图书馆CIP数据核字(2011)第004260号

基于模型的设计——MCU篇

刘 杰 翁公羽 周宇博 著

责任编辑 刘 晨

北京航空航天大学出版社出版发行

北京市海淀区学院路37号(邮编100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真 (010)82328026

读者信箱：emsbook@gmail.com 邮购电话：(010)82316936

北京时代华都印刷有限公司印装 各地书店经销

开本：787×1 092 1/16 印张：32 字数：819千字

2011年1月第1版 2011年1月第1次印刷 印数：4 000册

ISBN 978-7-5124-0315-4 定价：69.00元

前言

20 世纪 90 年代初，航空航天、汽车等行业中开始大量使用微控制器单元，开发人员首先发现通过建模与仿真，可以大大提高 MCU 系统的开发效率。到了 20 世纪 90 年代中期，出现了代码自动生成技术，使基于模型设计的雏形随之显现。

近十几年来，信息技术飞速发展，产品中软件控制代码呈爆炸性增长的趋势。一台中高档汽车的控制代码超过 500 万行，第四代战机 F－35 的软件代码则高达 1500 万行，它们都超过了人类第一次登月——阿波罗计划中软件代码的规模。随着代码量的迅速膨胀，传统的手工编程模式面临着产品开发周期被迫拉长，开发成本成倍增加，同时产品可靠性也难以保障等诸多难以克服的困难，已很难适应当今科技发展的需要，严重制约了我国现代化的进展。目前，世界上已有数家具有前瞻性的软件公司推出了自己的基于模型设计软件，如 SCADE、dSpace、MATLAB 等，它们是世界科技创新的技术源泉。空客 A380、美国通用公司的混合动力汽车、我国二汽新能源汽车的电池管理系统、喷气发动机、洛克-马丁公司的联合攻击机 F－35 等重大创新项目，都采用了基于模型的设计。

作者自 2000 年开始关注自动代码生成技术，以及后来的基于模型设计，尤其是最近三四年来，夜以继日地钻研基于模型设计的思想，阅读浏览了超过 10 万页的外文资料与技术文档，做了大量基于模型设计的实验，撰写了国内第一本基于模型设计的专著——《基于模型的设计及其嵌入式实现》。也许是外国政府或国际知名企业对这项新技术有所保留，网上很难找到有价值的开发实例，美国 CRC 出版社 2009 年 11 月出版的《Model-Based Design for Embedded Systems》一书，也仅是作了知识性的介绍。

通过这几年作者及实验室全体同仁的共同奋斗，基本掌握了基于模型的设计方法，并已将此技术推向了实用阶段。现将近年的研究成果整理成册，以期能为我国的科技创新，建立创新型国家贡献微薄之力。

考虑到国内有众多的 MATLAB 用户，本书以 MATLAB R2010b 为软件平台介绍基于模型设计的方法。书中涉及 8051 单片机、英飞凌 C166 单片机、Microchip dsPIC 数字信号控制器、ARM 处理器的基于模型设计的开发，其优势在于：在统一的可视化开发测试平台上，使用户轻松地从设计概念到实现一气呵成，减少不必要的重复劳动，这将大大缩短项目的开发周期、减少资金投入、提高代码的稳健性与一致性。一些对安全性要求不高的产品几周就可以完成，有效地规避了潜在的市场风险，使企业在激烈的市场竞争中占得先机；满足 DO-178b 航空电子规范的工作流程，使基于模型设计能提供解决那些对于安全性、可靠性要求极高甚至近乎苛刻的设计要求的绝好方案。

全书可分三个部分：1～4 章为第一部分，介绍模型的创建与调试，是基于模型设计的起点与核心；5～8 章为第二部分，介绍基于模型的 8051 单片机、英飞凌 C166 单片机、Microchip dsPIC3x 数字信号控制器、ARM 处理器的快速开发；第 9 章为第三部分，以直流电动机的 PID

目 录



第1章 MATLAB编程基础

MATLAB 是一种高度集成化的交互式编程环境，大到航空航天，小到计算二元矩阵的逆矩阵，都可以看到 MATLAB 的身影，其优势在于算法研究、数据分析与可视化、并行计算等方面。随着信息技术和计算机软硬件的不断发展，MATLAB 的应用领域得到了广泛的拓展，主要包括电机控制、飞行建模、音视频处理、通讯、测试与测量、财务建模分与分析等。

MATLAB 是基于模型设计的起点，本章仅对新版 MATLAB 软件的一些特色功能、MATLAB 程序的编写及调试方法、Embedded MATLAB 的编程规范作简单介绍，更详细的内容请读者参考 MathWorks 公司的相关技术手册及各种著作。

本章的主要内容如下：

- MATLAB 开发环境新功能。
- M 文件编写、调试与发布。
- Embedded MATLAB。

1.1 MATLAB R2010a 与 2010b 的若干更新

MATLAB R2010a 与 R2010b 针对 MATLAB 和 Simulink 新增加了若干功能，并对其他多款产品进行了更新和缺陷修复，本章首先列出 MATLAB 新增加及加强的功能。

1. 开发环境(MATLAB 7.10)

- 新增了解压缩功能，能够自动压缩和解压当前文件夹的文件和文件夹。
- 提示"当前文件夹"是否包含在 MATLAB 搜索路径列表。
- MATLAB 变量编辑器中的表格填写功能，可支持局部变量、子函数和嵌套函数。
- 扩展了曲线拟合、滤波器设计、图像处理与信号处理工具箱绘图选择界面的图形访问能力。
- 使用比较工具比较文件时，可高亮显示各行的变化；比较文件夹时，可按名称、类型、大小或时间排列比较结果。

2. 开发环境(MATLAB 7.11)

- 编辑器可高亮显示变量或子函数的所有使用情况,并识别共享变量。
- 可以将 ZIP 文件看作文件夹进行操作。
- 预览当前文件夹的图像文件内容,并以星号提示 MATLAB 文件是否保存。
- 在绘图选择界面中可以访问系统标识、制图和生物信息学工具箱的图形。
- 使用比较工具能够比较 ZIP 文件、文件夹和 Simulink 表单,并改善了 MAT 文件比较性能。

3. 语言和编程(MATLAB 7.11)

- 可使用若干组已命名的值,自行定义枚举数据类型。

4. 数学(MATLAB 7.11)

- 支持 64 位整型数据的算法。

5. 文件 I/O 和外部接口连接(MATLAB 7.11)

- 新增了 VideoWriter 对象,可创建大于 2GB 的 MPEG 和非压缩 AVI 文件。
- 支持 netCDF0.1,可将 HDF5 用作 netCDF API 的数据存储层。
- 加强了与 Microsoft .NET framework 的接口,支持与 Microsoft Office 产品的授权和交互。

根据本书的定位,以下选取几个开发环境典型的新功能,作简要说明。

1.1.1 压缩文件

为了备份文件、节省存储空间、与他人共享,通常是使用第三方压缩程序将文件打包压缩,而 MATLAB 7.10 集成了压缩功能。在目录浏览器可以直接建立或解压 ZIP 格式的压缩文件。MATLAB 7.11 进而又可将 ZIP 文件看作文件夹进行操作。

1. 创建压缩文件

选中需要打包的文件,选择右键菜单项 Creat Zip File,如图 1.1.1 所示。在当前目录建立以 Untitled1 为名的 ZIP 压缩文件,如图 1.1.2 所示。

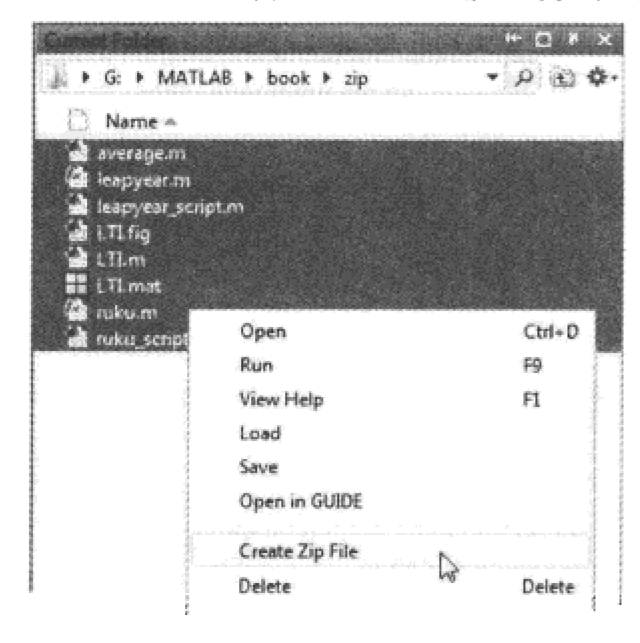


图 1.1.1 创建压缩文件

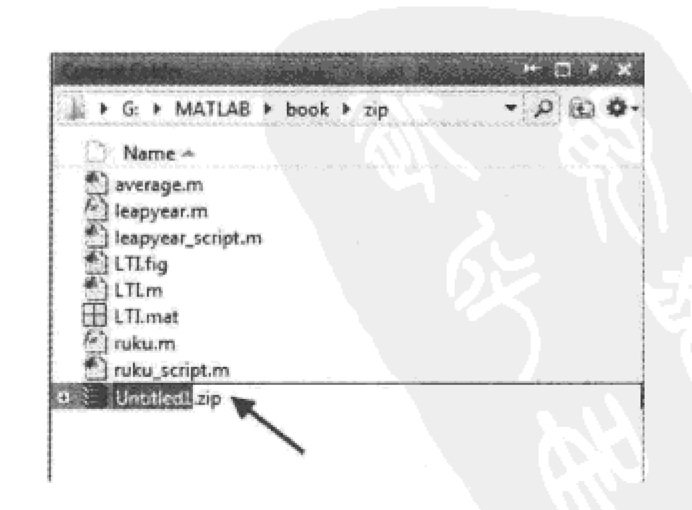
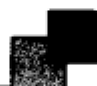


图 1.1.2 命名压缩文件



2. 压缩文件管理

单击压缩包左侧的+号，将其看作一个文件夹展开显示，如图 1.1.3 所示。双击压缩包内的各文件，可执行对应的操作，例如打开 M 文件，加载 MAT 数据等。不过这些文件是只读的，用户修改后需要将其另存。

3. 解压缩文件

双击压缩文件、选择压缩文件右键菜单项 Extract、或选中文件后按 Enter 键，在当前目录解压得到同名文件夹，如图 1.1.4 所示。当然，如果压缩包里包含与当前目录下同名的文件或文件夹，在解压时系统会提示用户是否替换当前文件。

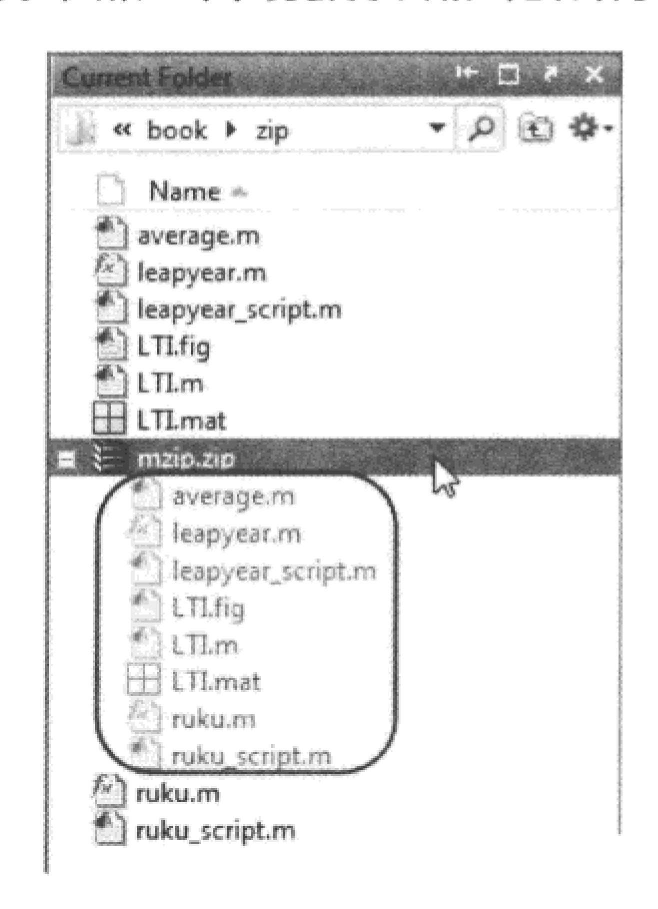



图 1.1.3　查看压缩文件

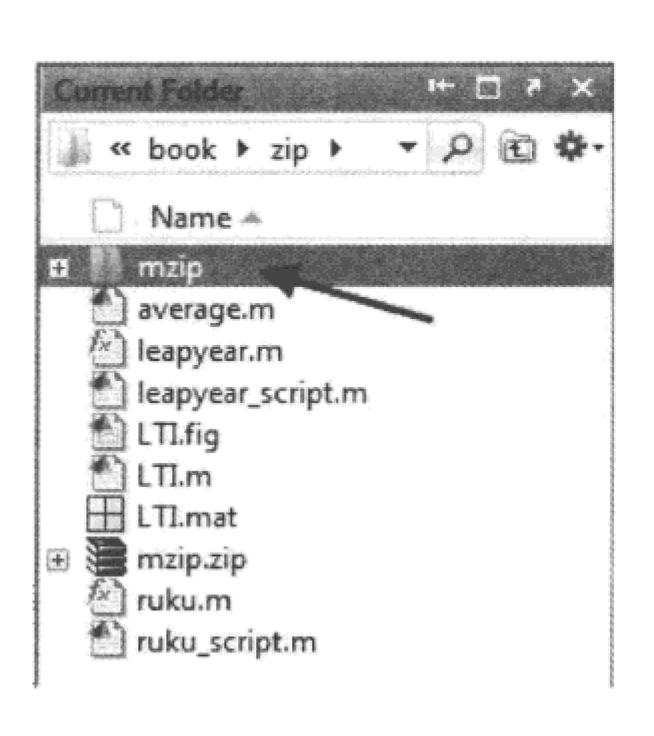


图 1.1.4　解压缩文件

1.1.2　目录浏览器

对于大型的工程，脚本、函数、数据等文件可能处于不同的文件夹，如果 a 目录下的脚本文件需要调用 b 目录下的函数与 c 目录下的数据，用户可以将 b、c 目录添加到 MATLAB 搜索路径，这类似于 C 语言的头文件。

单击 MATLAB 主窗口菜单项 File→Set Path...，打开搜索路径对话框，用户可使用左侧的按钮增加或删除搜索路径并调整路径顺序，如图 1.1.5 所示。

使用早期版本的用户可能感觉到，访问位于不同文件夹下的文件是个很麻烦的过程，需要频繁地切换目录。自 MATLAB 7.9 起，目录浏览器改进了显示方式，使用类似于 Windows 资源管理器的树形结构，用户可同时查看多个目录下的文件。MATLAB 7.10 将搜索路径体现在目录浏览器，淡化显示未添加入搜索路径的文件夹及下属文件。

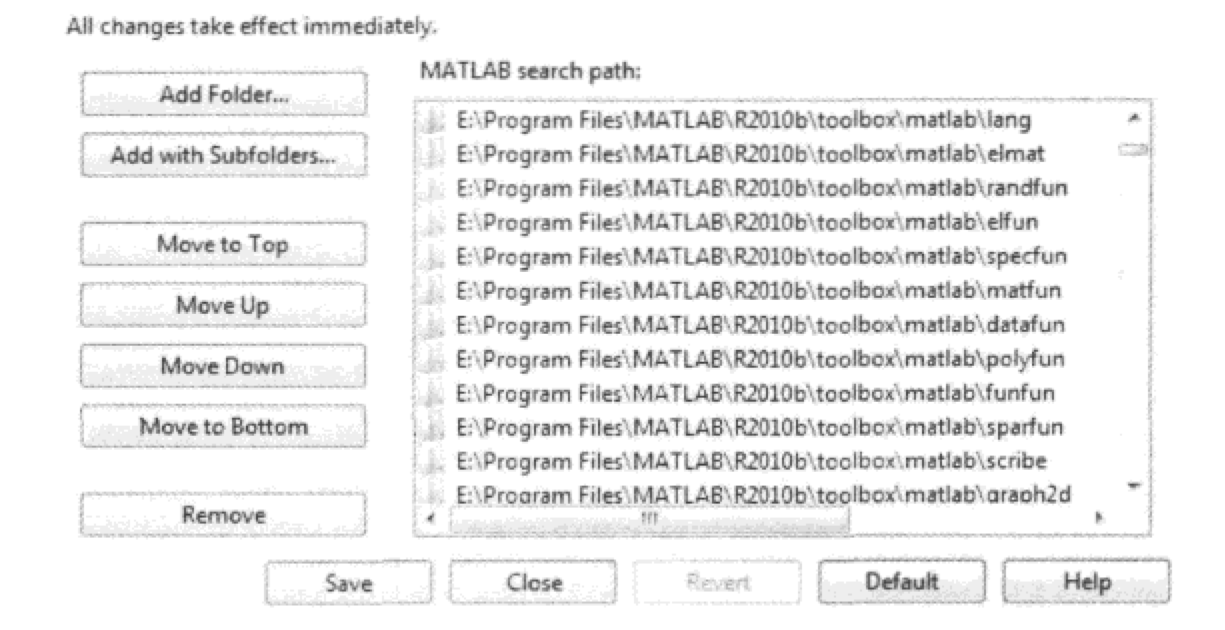


图 1.1.5 MATLAB 搜索路径

用户选中某文件夹，选择右键菜单项 Add to Path(或 Remove from Path)，将当前文件夹(或及其子文件夹)加入搜索路径(或移除)，如图 1.1.6 所示。



图 1.1.6 添加搜索路径

默认情况下，加入搜索路径的文件夹以正常的亮度显示，未加入搜索路径的文件夹则淡化显示，如图 1.1.7 所示。这样用户可明确地分辨当前目录下哪些文件夹及文件是可访问的，从而简化了搜索路径的设置。

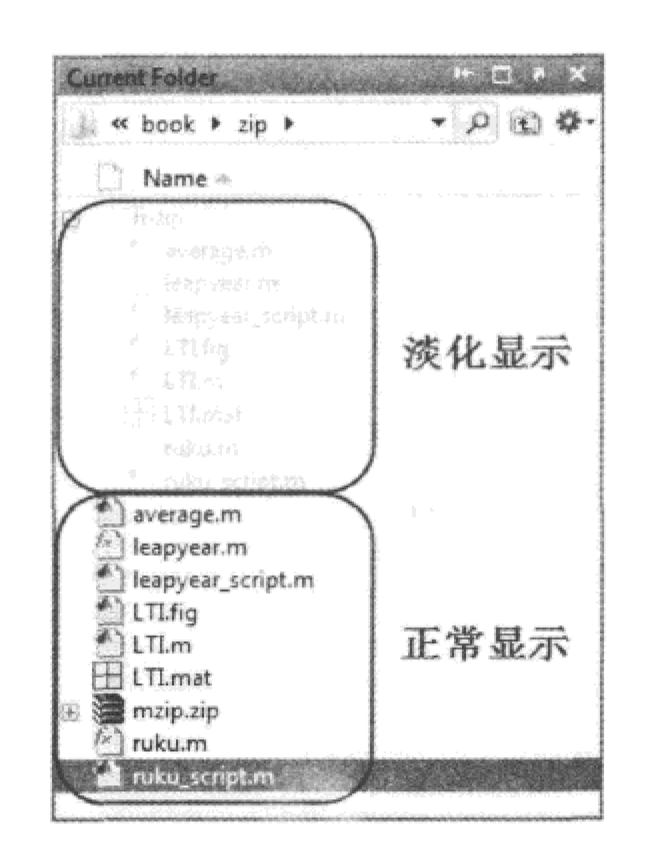
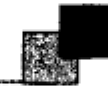


图 1.1.7 文件夹显示



单击 MATLAB 主窗口菜单项 File→Preferences...，选择 Current Folder 面板，其中复选项 Indicate inaccessible files 的功能与图 1.1.6 所示的右键菜单项 Indicate files not on path 一致，可开启或关闭淡化显示功能。拖动 Text and icon transparency 滑块，调整淡化显示的透明程度，如图 1.1.8 所示。

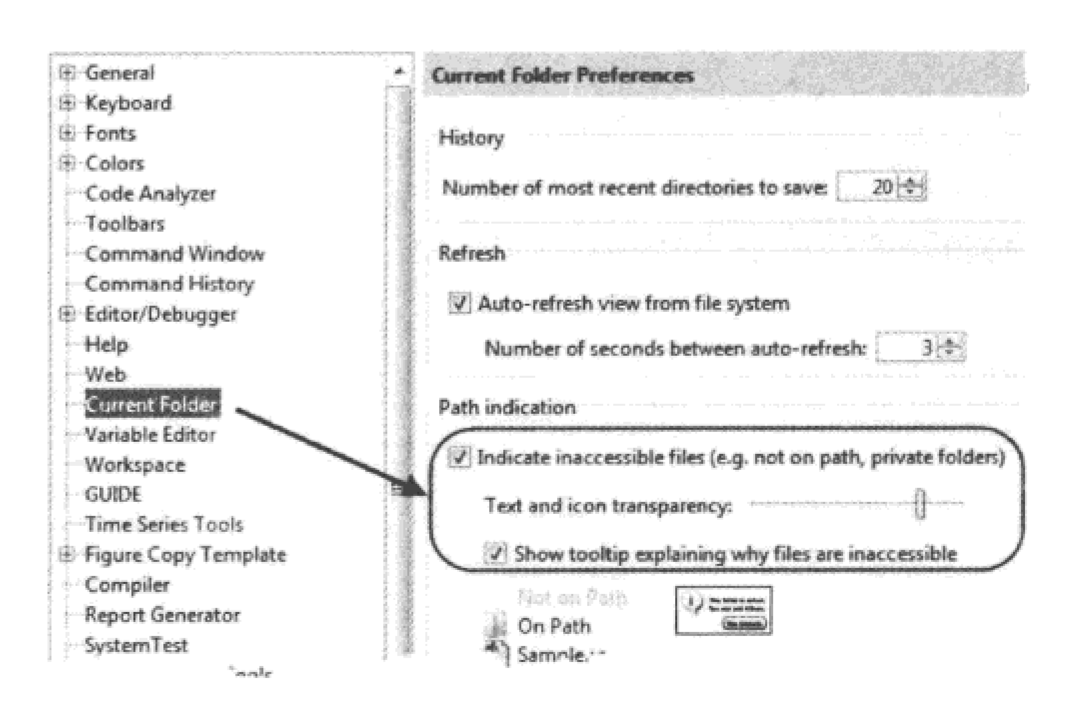
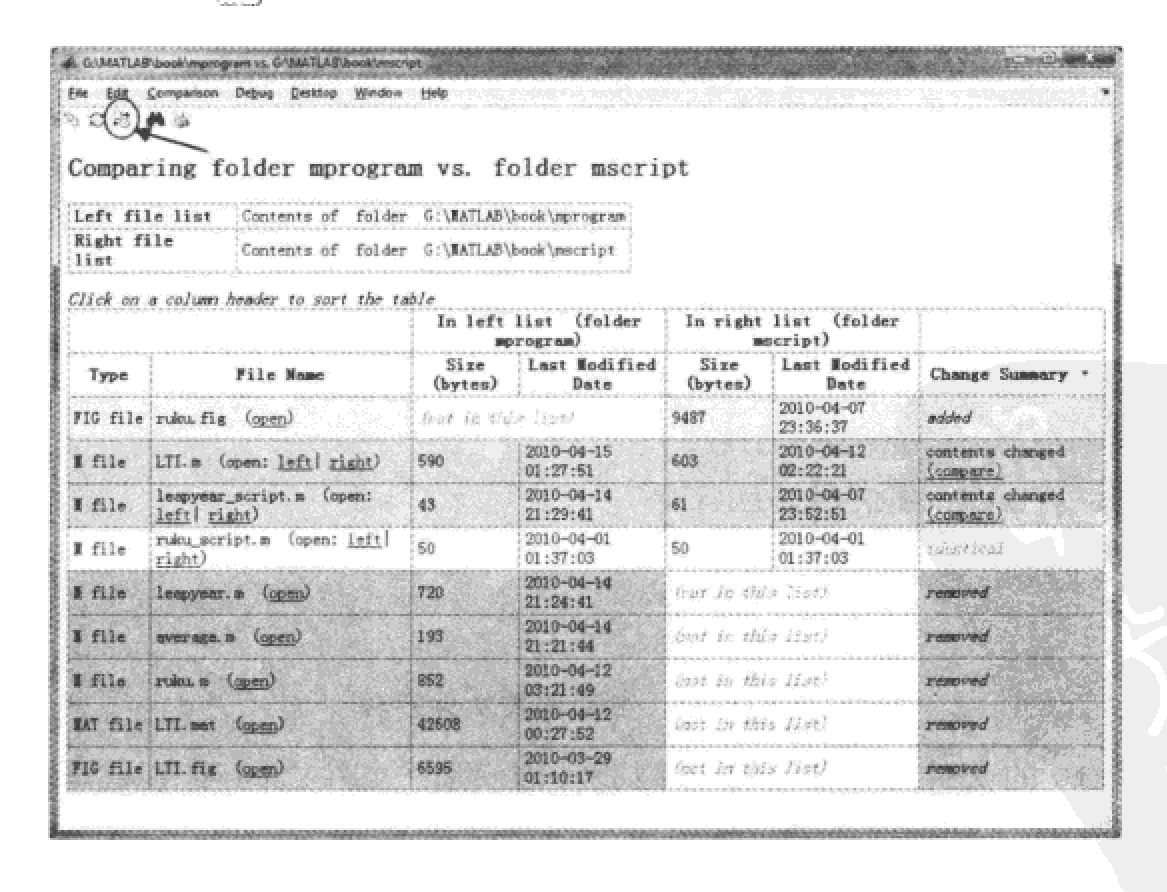


图 1.1.8　调整透明度

1.1.3　文件夹及文件比较

用户可以使用 MATLAB 文件比较工具，比较两个文件夹下属文件的类型、文件名、大小、修改时间，以及两个具体的文本文件、MAT 文件、二进制文件之间的差异。

绿色、红色、无色、蓝色分别表示右侧文件夹与左侧文件夹的比较结果：增加、修改、一致、删除。单击工具栏的按钮，可以调换两个文件夹位置，如图 1.1.9 所示。



		In left list (folder mprogram)		In right list (folder mscript)		
Type	File Name	Size (bytes)	Last Modified Date	Size (bytes)	Last Modified Date	Change Summary
FIG file	ruku.fig (open)	(not in this list)		9487	2010-04-07 23:36:37	added
M file	LTI.m (open: left| right)	590	2010-04-15 01:27:51	603	2010-04-12 02:22:21	contents changed (compare)
M file	leapyear_script.m (open: left| right)	43	2010-04-14 21:29:41	61	2010-04-07 23:52:51	contents changed (compare)
M file	ruku_script.m (open: left| right)	50	2010-04-01 01:37:03	50	2010-04-01 01:37:03	identical
M file	leapyear.m (open)	720	2010-04-14 21:24:41	(not in this list)		removed
M file	average.m (open)	193	2010-04-14 21:21:44	(not in this list)		removed
M file	ruku.m (open)	852	2010-04-12 03:21:49	(not in this list)		removed
MAT file	LTI.mat (open)	42608	2010-04-12 00:27:52	(not in this list)		removed
FIG file	LTI.fig (open)	6595	2010-03-29 01:10:17	(not in this list)		removed

图 1.1.9　比较结果

单击 Type、File Name、Size、Last Modified Date、Change Summary 等栏目名，可调整表格的排序方式(如图 1.1.9 按比较结果排序)。

单击链接 left 或 right，可以打开左侧或右侧文件夹的对应文件，若同时打开了两个文件，用户还可以将 MATLAB 编辑器窗口调整为纵向并列，具体分析文件的差异如图 1.1.10 所示。

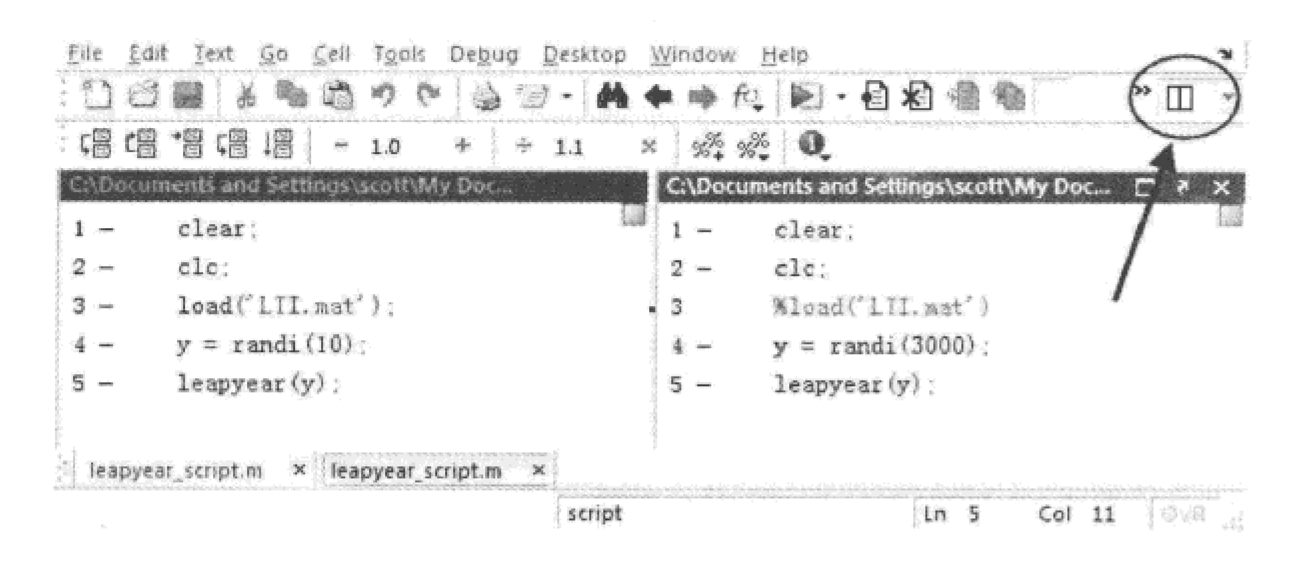


图 1.1.10　并列 MATLAB 编辑器窗口

显然人工比较容易出错，用户可继续单击图 1.1.9 表格最后一列的链接 compare，自动比较文件差异，如图 1.1.11 所示。

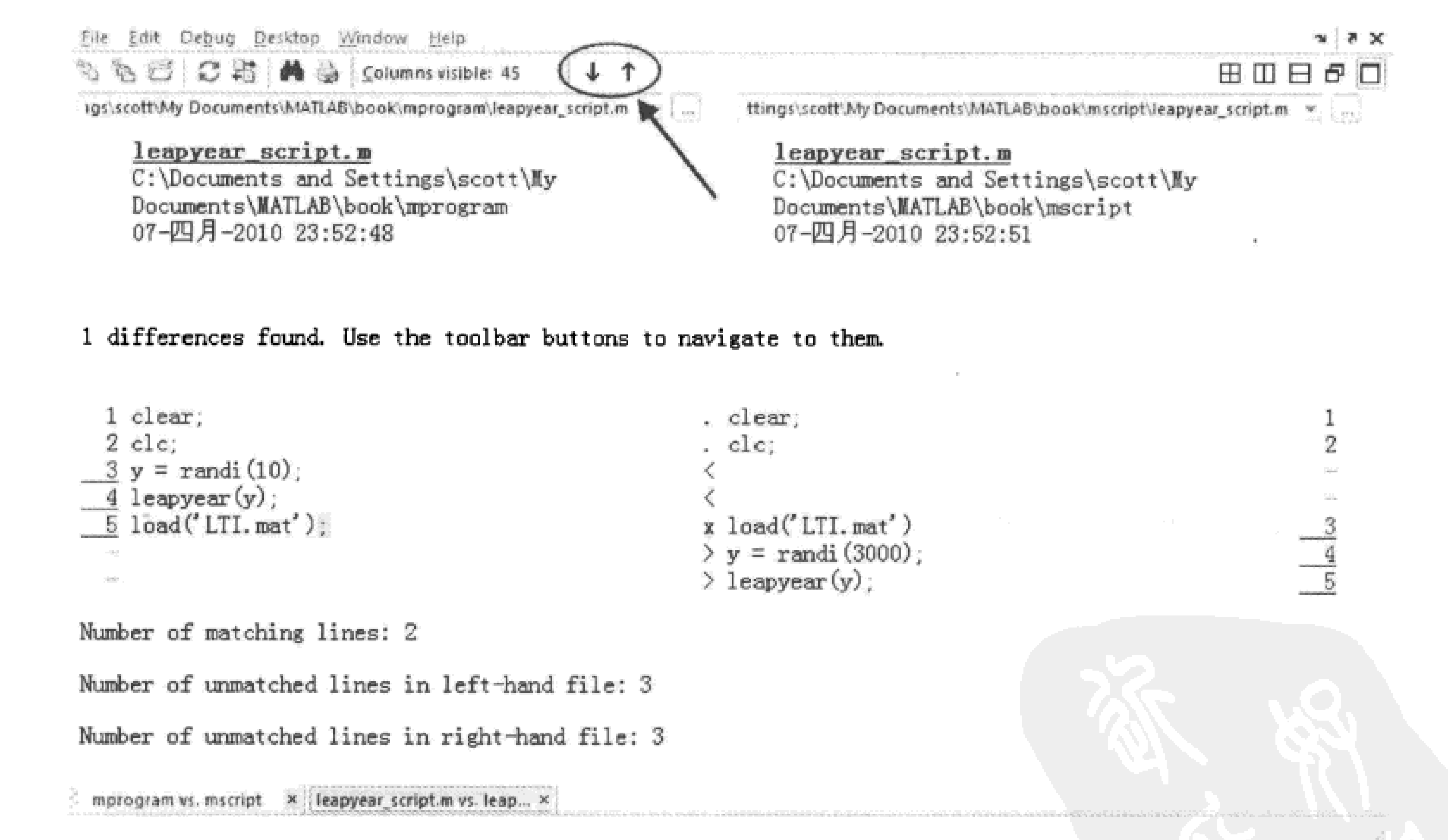


图 1.1.11　自动比较文件差异

MATLAB 7.10 的文件比较器新增了高亮功能，两个文件之间存在差异的行将被高亮显示，同时在工具栏新增了按钮[↓]与[↑]，便于逐一比较各个差异。



1.1.4 登录MATLAB文件交换服务器

在MATLAB环境中,用户可轻松地登录MATLAB File Exchange服务器,享受MATLAB文件交换服务。即用户可以使用由MathWorks公司或世界各地用户上传至MathWorks服务器的各种MATLAB程序、Simulink模型、演示视频等。合理有效地利用这些文件,可节省用户的学习与开发时间、启发思路、扩展视野。

使用该项服务是免费的,不过用户需要拥有MathWorks账户并能够访问互联网,当然建立账户也是免费的。

单击MATLAB主窗口菜单项Desktop → File Exchange,打开登录界面,如图1.1.12所示。

图1.1.12 用户登录

用户可以单击链接Creat an account创建一个Mathworks账户,或使用已创建的账户登录MATLAB File Exchange服务器,如图1.1.13所示。

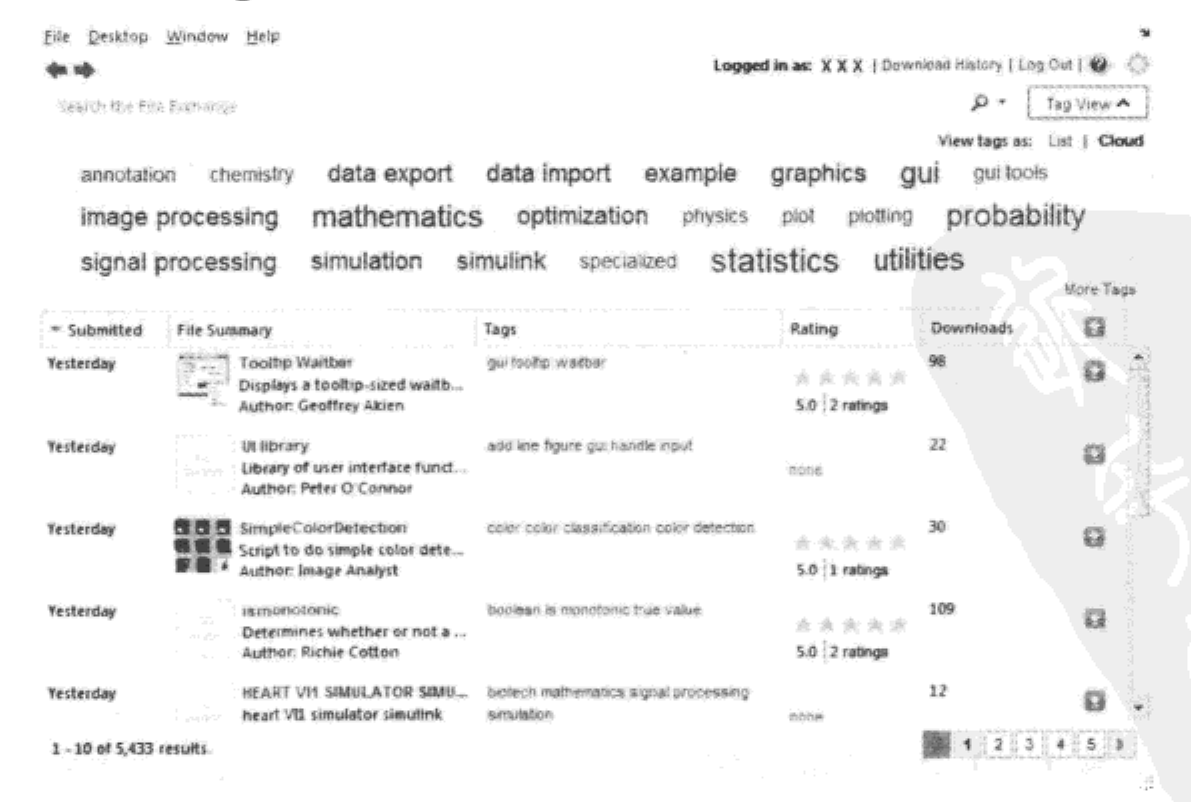


图1.1.13 MATLAB File Exchange主页

在搜索栏输入关键词，例如 BER，打开任意一个搜索结果，单击左上角 Download 按钮右侧的下箭头，可以选择下载到默认目录、当前目录或任意目录，如图 1.1.14 所示。

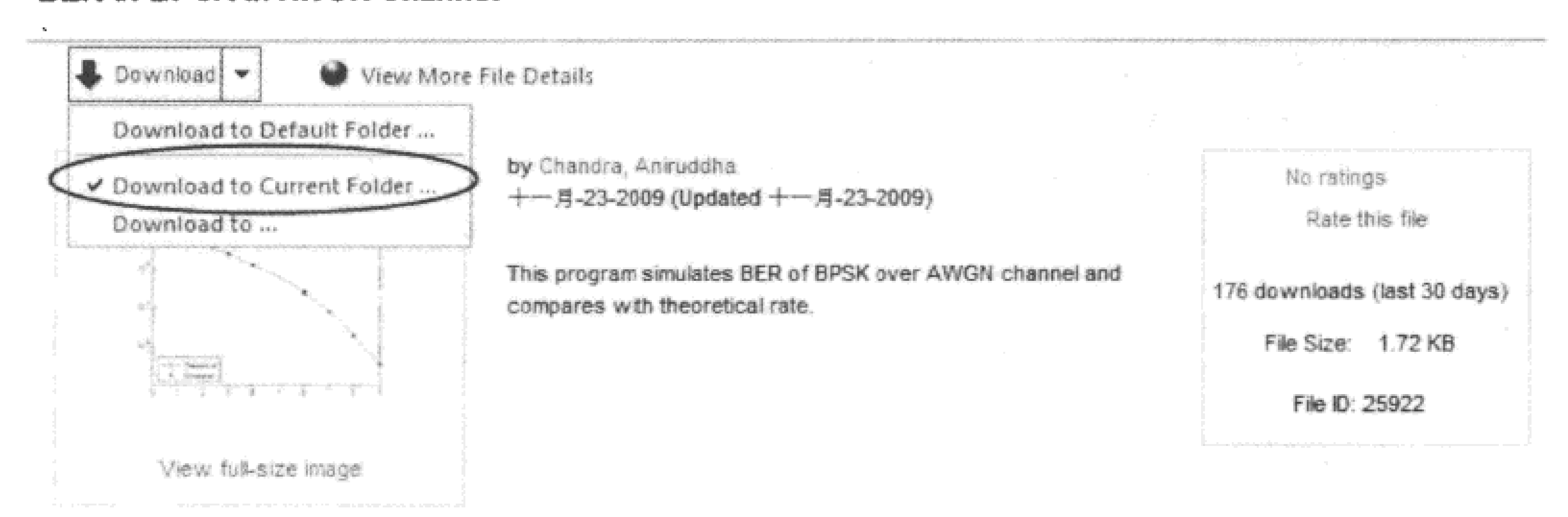


图 1.1.14　下载页面

与 R2009b 不同的是，由于 R2010a 增加了解压缩功能，下载的文件将自动解压缩到当前目录，不再是单独的 ZIP 文件。

1.2　M 文件的编写

MATLAB 有两种形式的 M 文件：M 函数文件与 M 脚本文件。编写 M 文件的好处在于用户可以将需要执行的一系列 MATLAB 指令包含在一个文件里，而在命令行窗口用户只需要输入一条执行该文件的指令。

完整的 M 文件应包含以下各部分，当然其中的函数定义行只用于 M 函数文件。

【例 1.2.1】

```
function y = fact(x)                          % 函数定义行
 % Compute a factorial value.                 % H1 行
 % FACT(x) returns the factorial of x,        % 帮助
 % usually denoted by x!
 % Put simply, FACT(x) is PROD(1:x).          % 注释
y = prod(1:x);                                % 函数体/脚本体
```

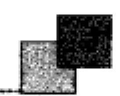
1.2.1　M 文件结构

1. 函数定义行

函数定义行必须在 M 函数文件第一行，它定义了函数的名称以及输入/输出参数的数量和顺序。

如例 1.2.1，function 是关键字，y 是输出参数，fact 是函数名，x 是输入参数。

对于多输入/输出的情况，将输出参数放入方括弧[ ]，输入参数放入圆括弧()，参数之间



用半角逗号区隔，如 function [x, y, z] = func(a, b, c)。

若函数没有输出，可将输出留空或使用一对空的方括弧。如 function printresults(x)或 function[ ]=printresults(x)。

函数名必须以字母开始，此后可以包含任何字母、数字或下画线。文件名长度不能超过最大允许长度 N。虽然理论上 M 函数名可以是任意长度的，但 MATLAB 仅截取前 N 个字符作为函数名，因此用户需要确保该函数名是唯一的。使用 namelengthmax 可以查看允许的最大文件名长度。

MATLAB 系统有一些保留字，用户不能使用它们作为函数名，使用 isvarname 命令检查某个函数名是否合法，如 isvarname myfun。返回值为 1 表示可用，返回值为 0 表示不可用。

M 函数的函数名一般也是该 M 函数文件的文件名，如果二者不一致，M 文件里的函数名将被忽略，用户应该使用 M 函数文件的文件名来调用该函数。

例如某函数文件名为 average. m，而文件内的函数命名为 computeAverage，则用户应该使用 average 来调用该函数。为避免混乱，通常应统一这两个名称。

2. H1 行

M 文件的简单描述以%开始。如例 1.2.1：% Compute a factorial value。

当用户在 MATLAB 命令窗口输入 help functionname 时，H1 行将在帮助信息的第一行显示。若使用 lookfor 命令查询，则仅显示该 H1 行。因此用户在编写 H1 行时必须简明扼要。

3. 帮　助

用户可以为自己编写的 M 文件增加一些联机帮助信息，这些信息可以是一行，也可连续多行的。MATLAB 系统认为紧随 H1 行后，一组连续以%开始的文字即该函数的联机帮助信息，一旦中断，即认为帮助信息结束。即使此后还有注释行，它们也将被忽略。

注意：用户 M 文件里的帮助信息只能在 MATLAB 命令行窗口显示，在 MATLAB 帮助浏览器中是看不到的。帮助浏览器只能用来查看 MathWorks 公司系列产品的帮助信息。

4. 注　释

注释文本以%开始，它可以插在 M 文件的任何位置，包括某一行代码的后面。另外用户可以在 M 文件的任意位置加入空行。尽管空行将被忽略，不过它可以用来结束帮助信息。

有时需要分行显示的注释，虽然可以在每行之前分别加上%，比较简便的方法是使用块注释标记 %{ 和 %} 。这两个符号必须独立占用一行，符号与注释行间不能有空行，如：

```
%{
This next block of code checks the number of inputs
passed in, makes sure that each input is a valid data
type, and then branches to start processing the data.
%}
```

5. 函数体/脚本体

函数体/脚本体可以包括函数调用、变量赋值、计算式、注释、空行、以及程序结构体，如流控制和交互输入/输出。

【例 1.2.2】 计算一组数值的平均值

```
[m,n] = size(x);                          % 数组维度
if (~m || (m == 1 && n == 1))             % 流控制
   error('Input must be a vector');       % 错误提示
end
y = sum(x)/length(x);                     % 平均值计算
```

1.2.2 M 脚本文件

脚本文件是最简单的 M 文件，它不需要输入参数，同时也不返回输出，常用于执行那些需要多次执行的指令。

脚本文件使用 MATLAB 的基本工作空间，可对已有的变量进行操作或者新建变量。脚本文件执行完毕，所有变量都将保留在基本工作空间，供后续使用。在命令行窗口输入 whos，可以查看这些变量。

显然，运行了脚本文件，此前保存在基本工作空间的同名变量都将被覆盖。

【例 1.2.3】 二阶网络的冲激响应与阶跃响应

二阶 RCL 网络如图 1.2.1 所示，其中 $R=200\ \Omega$、$L=10$ H 、$C=3000$ nF。

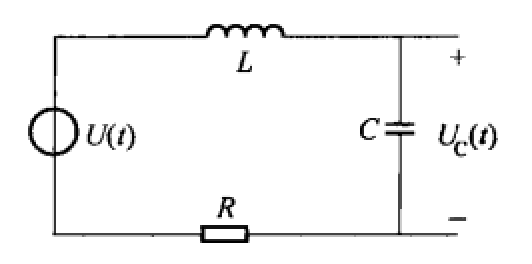


图 1.2.1 二阶 RCL 网络

根据基尔霍夫电流定律，有： $u(t)=L\dfrac{\mathrm{d}i}{\mathrm{d}t}+Ri+u_c$

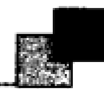
其中电容两端的电压为： $u_c=\dfrac{1}{C}\int i\mathrm{d}t$

两式各取拉普拉斯变换，整理得： $U(s)=(LCs^2+RCs+1)U_c(s)$

传递函数为： $H(s)=\dfrac{U(s)}{U_c(S)}=\dfrac{1}{LCs^2+RCs+1}=\dfrac{100}{3s^2+60s+100}$

为得到该二阶网络的冲激响应与阶跃响应，编写脚本如下：

```
clear;
clc;
num = 100;                      % 定义分子
den = [3,60,100];               % 定义分母
h = tf(num,den);                % 建立状态方程
s = ss(h);                      % 转化到状态空间
[yi,ti,xi] = impulse(s);        % 单位冲激响应
[ys,ts,xs] = step(s);           % 单位阶跃响应
plot(ti,yi,'-k');
```



```
hold on;
plot(ts,ys,'--r');
hold off;
xlabel('t');                        % X 轴标签
ylabel('U_c(t)');                   % Y 轴标签
axis([0 4 -0.5 1.5]);               % 设置 X,Y 轴数据显示范围
grid on;                            % 显示坐标网格
h = legend('impulse','step',1);             % 图例句柄
set(h,'Interpreter','none');                % 显示图例
```

保存上述代码并运行，结果显示如图 1.2.2 所示。

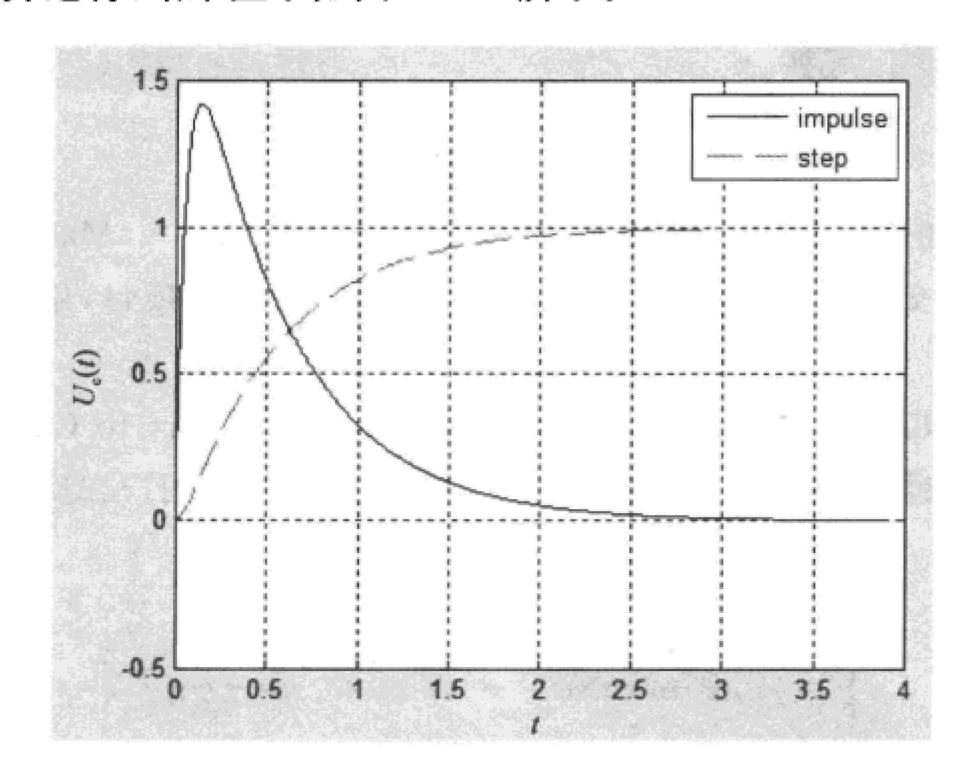
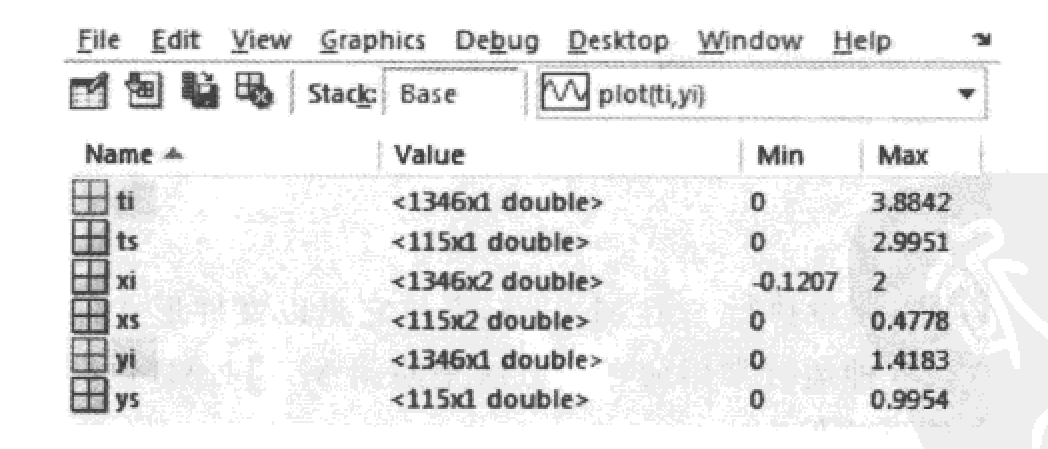


图 1.2.2　二阶 RCL 网络响应

对于简单的绘图，用户不必编写 M 脚本或在命令行窗口逐一输入代码，使用 MATLAB 工作空间浏览器上方的 Select data to plot 绘图按钮，可快速得到需要的图形。

设上例的变量已存在于工作空间，则依次选中变量 ti、yi，如图 1.2.3 所示。



File Edit View Graphics Debug Desktop Window Help

Stack: Base　plot(ti,yi)

Name	Value	Min	Max
ti	<1346x1 double>	0	3.8842
ts	<115x1 double>	0	2.9951
xi	<1346x2 double>	-0.1207	2
xs	<115x2 double>	0	0.4778
yi	<1346x1 double>	0	1.4183
ys	<115x1 double>	0	0.9954

图 1.2.3　选中变量

(1) 按窗口右上方的 plot(ti,yi) 按钮，即可得到与图 1.2.2 一致的结果。同时在命令行窗口显示对应的绘图代码，用户可将此作为后期编程的参考。

```
>> plot(ti,yi,'DisplayName','ti vs. yi','XDataSource','ti','YDataSource','yi');figure(gcf)
```

（2）按绘图按钮右侧的下拉箭头，可以得到更多的绘图项目，将鼠标指针悬停在某绘图项目，即显示该项目的帮助信息，如图 1.2.4 所示。

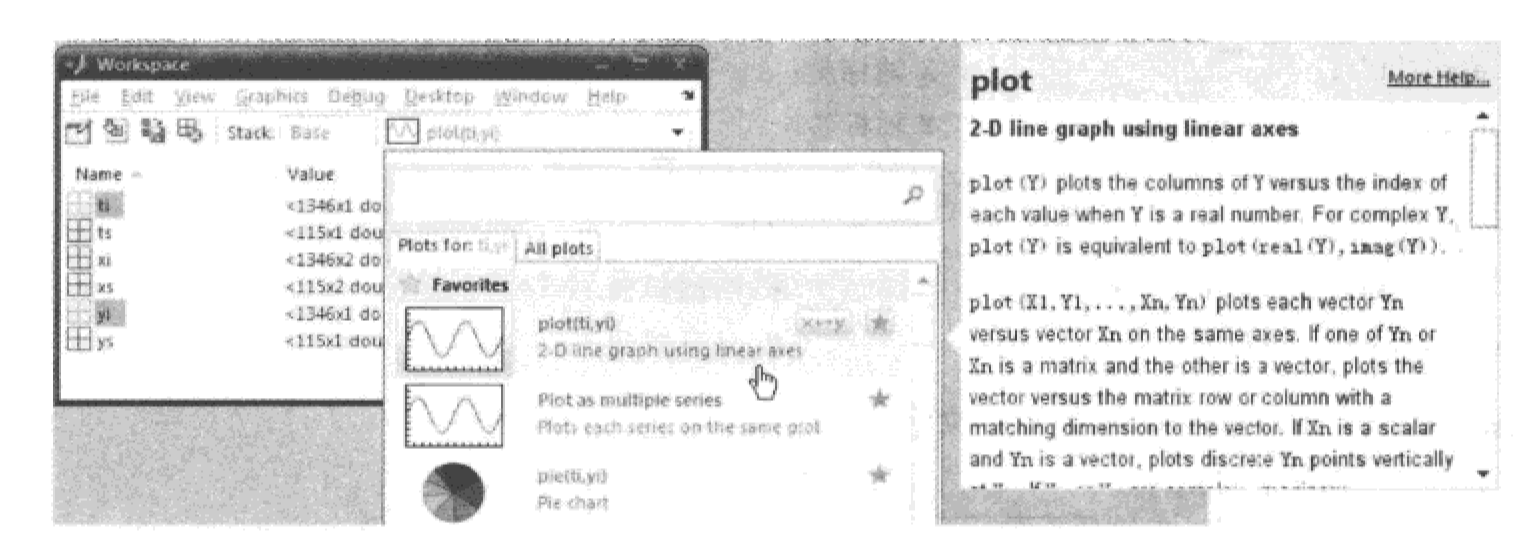


图 1.2.4　项目帮助信息

（3）在选择绘图变量时，系统默认先选中的为 x，后选中的为 y，Mathworks R2009b 提供了按钮 x↔y 允许用户交换 xy 变量，便于重新绘图。对于常用的项目，用户可以按下★，将其列入收藏。

（4）右击某一绘图项目，如图 1.2.5 所示，选择菜单项 Insert to Command Windows 或 Copy，可以直接将该绘图的命令插入命令行窗口或粘贴板，便于用户二次修改。

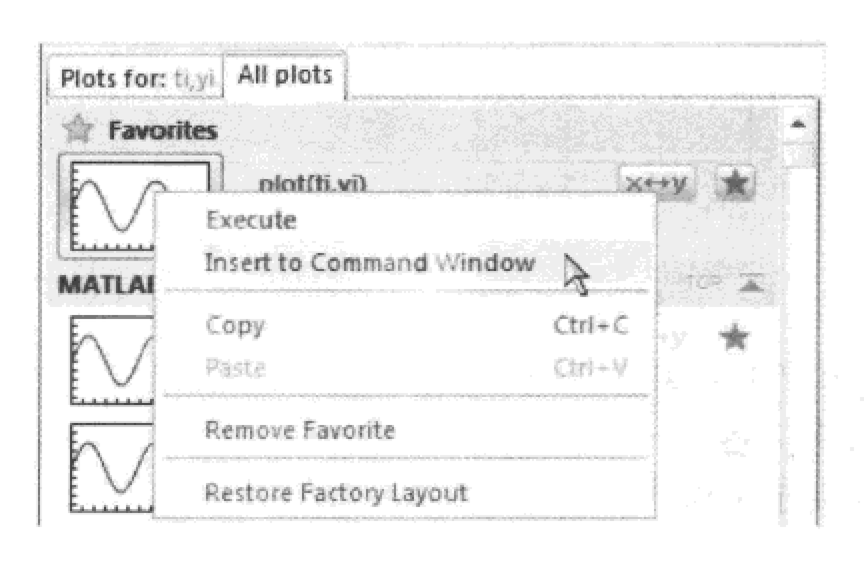


图 1.2.5　插入命令

1.2.3　快捷方式

用户可以使用 M 脚本批量执行一组 M 代码，不过它是以文件形式保存的，因此在使用之前必须先找到该文件，另一种运行常用命令的方法是使用 MATLAB 快捷方式，其功能类似于 M 脚本，但它并不以文件形式保存。

（1）单击 MATLAB 主窗口菜单项 Desktop→Toolbars→Shortcuts，开启或关闭主窗口的 Shortcuts 工具条，如图 1.2.6 所示。

（2）右击 Shortcuts 工具条，选择 New Shortcut 选项，如图 1.2.7 所示。





图 1.2.6 快捷方式工具条

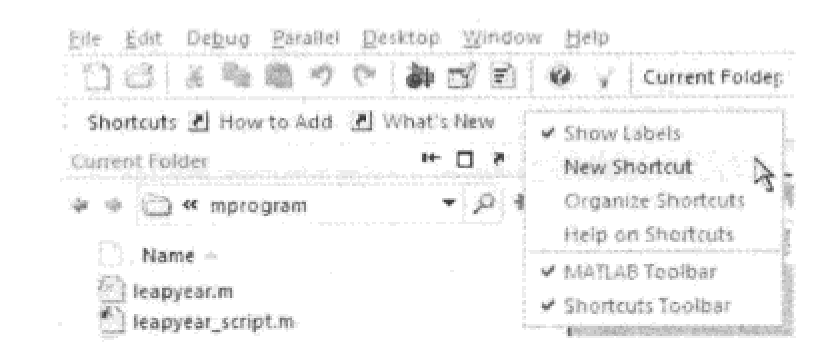


图 1.2.7 新建快捷方式菜单

按图 1.2.8 所示的编辑窗口进行设置并保存，在 Shortcuts 工具条即得到一个清空命令行窗口的快捷方式，如图 1.2.9 所示。

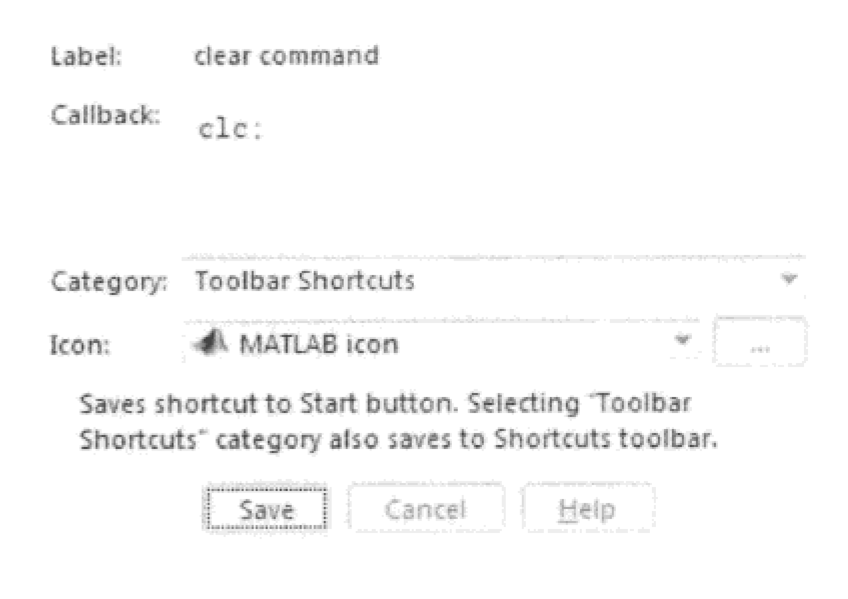


图 1.2.8 设置快捷方式

图 1.2.9 新建快捷方式

(3) 用户也可以在命令行窗口、命令历史窗口或 M 文件窗口，选中一系列代码，拖入 Shortcuts 工具条，在后续跳出的编辑窗口，输入显示标签等信息，建立快捷方式，如图 1.2.10 所示。

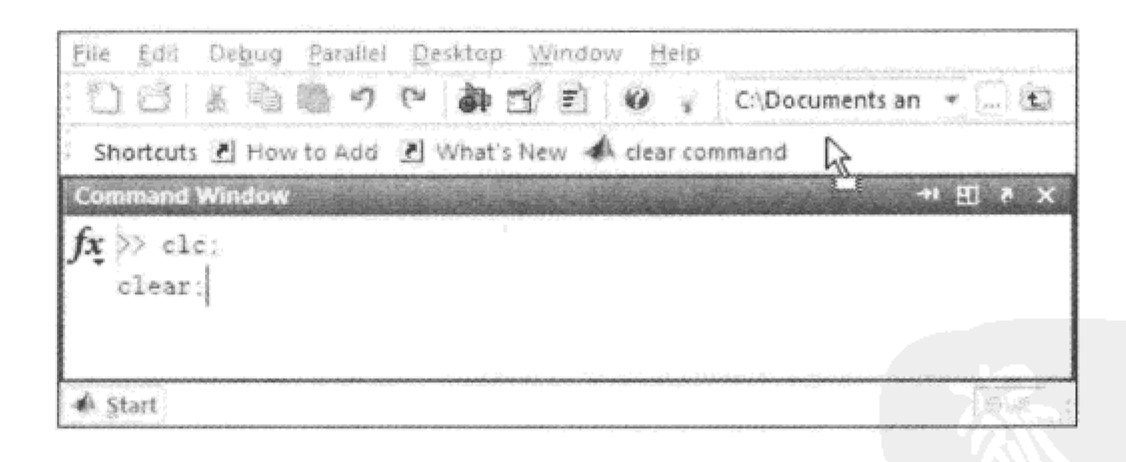


图 1.2.10 快速新建快捷方式

1.2.4 M 函数

M 函数文件可接受输入参数，并返回输出。如上文所述，使用 M 文件定义 M 函数，该文件必须以函数定义行作为起始行，或以 end 为结束，或以另一个函数的定义行作为上一个函数

的结束。但如果用户在一个函数体内定义了一个或多个嵌套函数，则必须使用 end 结束某嵌套函数。

每个 M 函数使用各自的工作空间，可称为函数工作空间，独立于人们平常通过命令行或脚本文件访问的 MATLAB 基本工作空间。

M 函数可分为以下几类：

(1) 主函数(Primary M-File Functions)，它是每个 M 函数文件中的第一个函数，通常它包含的是主程序。

(2) 子函数(Subfunctions)，M 函数文件可以包含多个函数，除了主函数，其他函数即称为子函数。

(3) 嵌套函数(Nested Functions)，它定义在另一个函数内，可以增加函数的可读性与灵活性。

(4) 匿名函数(Anonymous Functions)，一种可快速定义的函数，它可以出现在另一个函数内，更重要的是可以在 MATLAB 命令行定义匿名函数。

(5) 重载函数(Overloaded Functions)，特别适合于用户需要建立一个输入参数能够接受不同类型数据的函数。这类似于其他面向对象语言中的重载函数。

(6) 私有函数(Private Functions)，它只能由处于其父目录的 M 函数访问。主函数、子函数、嵌套函数、私有函数之间的区别仅仅是函数的地位，函数结构没有明显不同，众多文献已有详细论述，本文不再赘述。

【例 1.2.4】 闰年函数

根据历法，公历闰年精确的计算方法如下：

(1) 普通年能被 4 整除的为闰年。

(2) 世纪年能被 100 整除而不能被 400 整除的不是闰年。

(3) 对于数值很大的年份能整除 3200，但同时又能整除 172 800 的是闰年。

为简化说明过程，本函数不用于判断数值很大的年份。

```
function str = leapyear( y )
% 判断一个年份是否为闰年
  if rem(y,4)== 0                 % 年份能被 4 整除
    if rem(y,100)== 0             % 是否是整 100 年
      if rem(y,400)== 0           % 整 100 年能否被 400 整除
         leap = 1;                % 整 400 年，是闰年
      else
         leap = 0;                % 整 100 年但非整 400 年，非闰年
      end
    else
       leap = 1;                  % 被 4 整除但非整 100 年，是闰年
    end
  else
     leap = 0;                    % 不能被 4 整除，非闰年
  end
%% 输出结果
```



```
  if leap == 1
    str = strcat([num2str(y),'IS a leap year.']);
                                        % 若 leap 为真,输出肯定字符串
  else
    str = strcat([num2str(y),'is NOT a leap year.']);
                                        % 若 leap 为真,输出否定字符串
  end
end
```

在 MATLAB 命令行调用该函数,得到:

```
>> y = randi(3000)
y = 2725
>> leapyear(y)
ans = 2725 is NOT a leap year.
```

【例 1.2.5】 使用 Runge-Kutta 法求解一阶常微分方程

一阶常微分方程的初值问题表示如下:

$$\begin{cases} \dfrac{dy}{dx}=f(x,y) \quad x\in[a,b] \\ y(a)=y_0 \end{cases}$$

只要 $f(x,y)$在$[a,b]\times R^1$ 上连续,且关于 y 满足利普希茨(Lipschitz)条件,则上述问题存在唯一解,因此可以使用 Runge-Kutta 算法得到其数值解。

四阶 Runge-Kutta 算法表示如下:

$$\begin{cases} y_{n+1}=y_n+\dfrac{h}{6}(k_1+2k_2+2k_3+k_4) \\ k_1=f(x_n,y_n) \\ k_2=f\left(x_n+\dfrac{1}{2}h,y_n+\dfrac{h}{2}k_1\right) \\ k_3=f\left(x_n+\dfrac{1}{2}h,y_n+\dfrac{h}{2}k_2\right) \\ k_4=f(x_n+h,y_n+hk_3) \end{cases}$$

使用 M 函数实现的代码如下:

```
function y = ruku(fx,xmin,xmax,y0)
 % 采用四阶龙格库塔法数值积分
 % fx 为函数名
 % x0 为自变量的初始值
 % y0 为因变量的初始值
h = 0.01;               % 步长
x = xmin:h:xmax;        % 自变量
n = length(x);          % 计算点数
y = zeros(1,n);         % 初始化 y 向量
y(1) = y0;              % y 初值
 % % 四阶龙格库塔法
```

```
for i = 1:n
    k1 = feval(fx,x(i),y(i));
    %% k2
    x_temp = x(i) + h/2;
    y_temp = y(i) + k1 * h/2;
    k2 = feval(fx,x_temp,y_temp);
    %% k3
    x_temp = x(i) + h/2;
    y_temp = y(i) + k2 * h/2;
    k3 = feval(fx,x_temp,y_temp);
    %% k4
    x_temp = x(i) + h/2;
    y_temp = y(i) + k3 * h/2;
    k4 = feval(fx,x_temp,y_temp);
    %% y(n+1)
    y(i+1) = y(i) + (k1 + 2 * k2 + 2 * k3 + k4) * h/6;
end
%% 绘图
y = y(1:n);
plot(x,y);
title('4 order Runge - Kutta');
xlabel('x');              % X 轴标签
ylabel('y');              % Y 轴标签
grid on;
end
```

在 MATLAB 命令行调用该函数，运行结果如图 1.2.11 所示。

```
>> fx = @(x,y) x^2 - y;
>> ruku(fx, - 3,3,2);
```

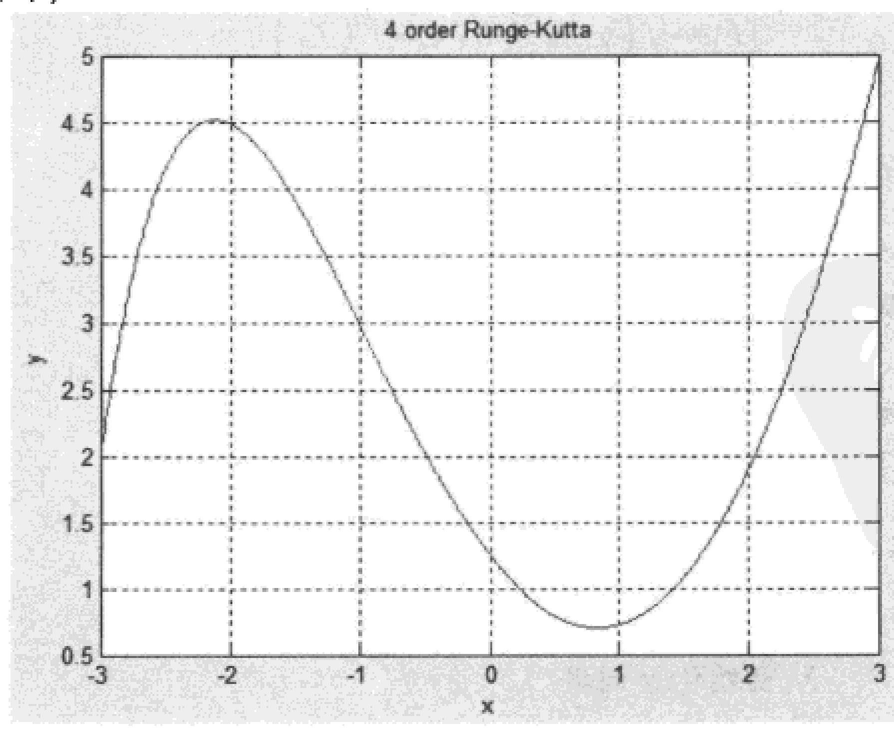
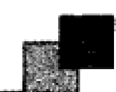


图 1.2.11　一阶常微分方程数值解



1.2.5　匿名函数

例 1.2.5 使用到了匿名函数 fx = @(x,y) x^2－y。这是一种简单的 M 函数，不需要为它创建 M 文件，可以在命令行窗口直接定义，也可以在 M 函数文件或脚本文件里定义。

建立匿名函数的指令是：fhandle = @(arglist) expr

其中 expr 表示函数体，它可以是任意的单一合法的 MATLAB 表达式；arglist 表示输入参数（多个参数可用半角逗号区隔）。这两者与其他 M 函数没有区别。

操作符@的作用是建立一个函数句柄，用户可以使用句柄调用对应的匿名函数。此后利用指令 fhandle(arg1, arg2, ..., argN)即可运行该函数，得到对应的输出。

定义一个匿名函数，操作符@是必不可少的。

值得注意的是，函数句柄不仅仅用于访问匿名函数，用户也可以对任意的 MATLAB 函数定义句柄。不过指令格式有所不同：fhandle = @functionname（如 fhandle = @sin）。

【例 1.2.6】　单输入匿名函数

下面的指令建立了一个计算二次方的匿名函数：

```
>> sqr = @(x) x^2;
```

sqr 即这个匿名函数的句柄，可以这样计算一个数的平方：

```
>> a = sqr(5)
a = 25
```

【例 1.2.7】　双输入匿名函数

下面的指令建立了一个二元函数：

```
>> sumxy = @(x, y) (2 * x + 3 * y);
```

取 x = 3，y = 4，计算该二元函数值：

```
>> c = sumxy(3, 4)
c = 18
```

1.2.6　函数提示

1. 函数名提示

MATLAB 7.10 新增了函数提示功能，若用户仅依稀记得某函数的起始字母，可以在输入这些字母后按 Tab 键，系统自动列出以这些字母开始的所有函数，供用户选择，如图 1.2.12 所示。

2. 参数提示

用户若事先已输入某一函数的函数名，但不确定该函数的输入参数，则在函数名后加上一个左圆括弧，稍等片刻，系统即显示该函数的各种可能的表达式格式，如图 1.2.13 所示。

提示框的内容将根据用户后续的输入不断筛选显示，如图 1.2.14 所示。

单击提示框下部的链接 More Help...，可链接到帮助文档，如图 1.2.15 所示。

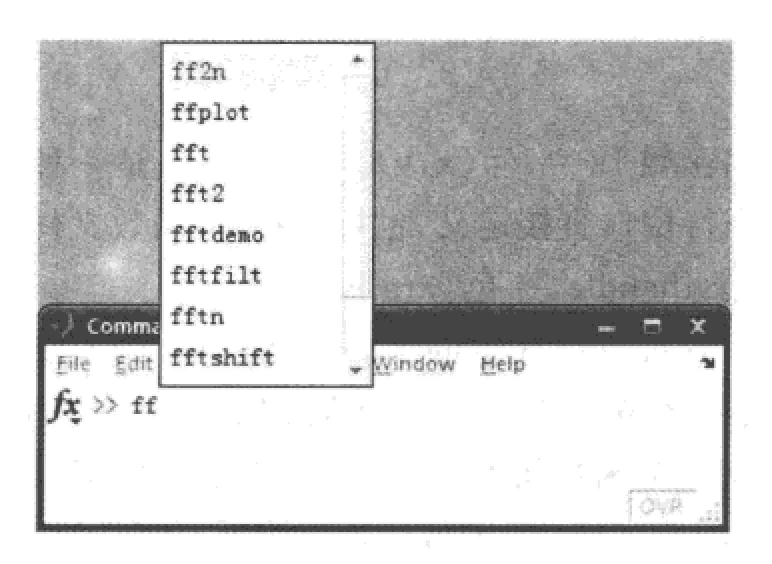


图 1.2.12 函数名提示



图 1.2.13 函数参数提示

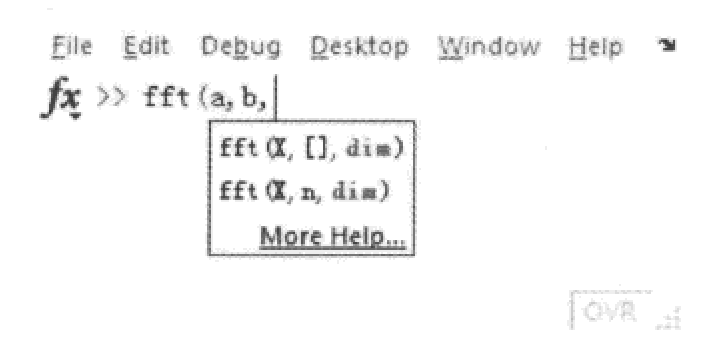


图 1.2.14 参数筛选显示

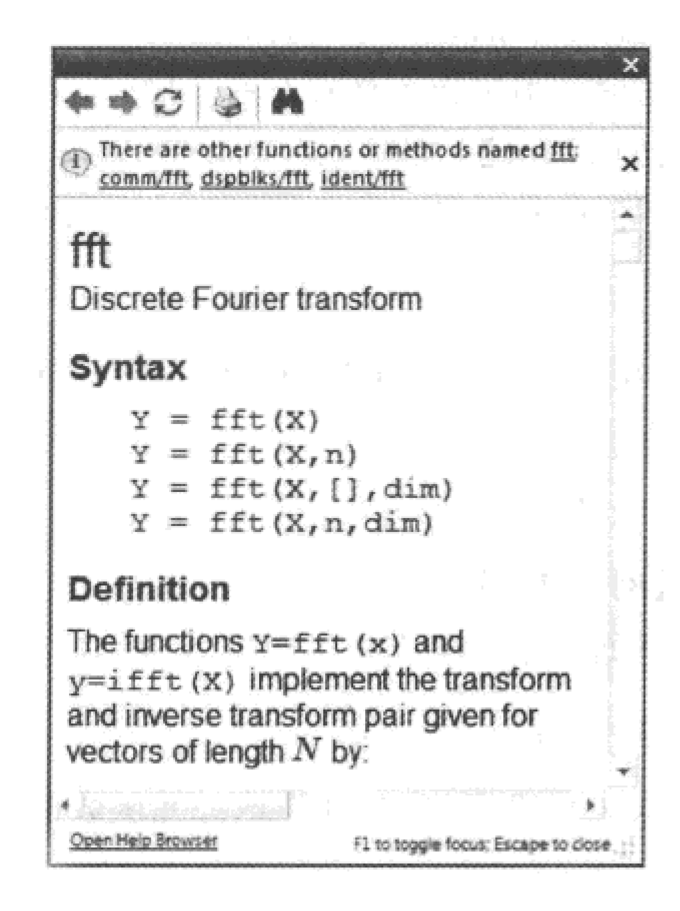
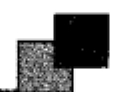


图 1.2.15 函数帮助

1.3 M 文件的调试

1.3.1 M-Lint

MATLAB 7.2 增加了 M-Lint 代码检查器，或直接称作 M-Lint，可以检查用户代码中存在的问题，并提出推荐的修改方法，使之性能与可靠性达到最大。MATLAB 7.10 将 M-Lint 称作 Code Analyzer，但函数 mlint 与 mlintrpt 依然保持原名。



用户可以在代码编辑过程中，根据 M-Lint 自动更新的提示信息，随时修改对应的代码。对于某些提示，M-Lint 还提供了额外的信息，或者自动修改功能，或者两者皆有。实时代码检查与修改界面的使用将在第 1.5 节 Embedded MATLAB 中介绍。

1.3.2　使用 cells 加快调试

M 文件通常是由几个功能代码区组成的，对于一个很长的 M 文件，在某个时候用户只专注于调试某个代码区。自 MathWorks R14 起，用户可以使用 cell 模式加快 M 文件的调试。字面理解，cell 就是表示某个代码区，每个 cell 包含了若干行代码，用户可以将这些代码看作一个 M 脚本，对其进行单独调试。另外，定义了 cell 的 M 文件，其发布的文档更具有可读性。

cell 模式仅用于常规 M 文件，不适用于 Embedded MATLAB 与其他 TXT 文档。

1. 定义 cell

cell 是 M 文件里的一段代码，因此必须定义它的边界：这种边界分为用户定义的显性边界与 MATLAB 默认的隐性边界。

定义显性边界的一般方法是在 cell 的开始插入一个空行，并输入两个连续的百分号"%%"（图 1.3.1 行 1），称作显性 cell 边界符。当然边界符也可以插在该 cell 的第一行代码前，用空格与后面的代码隔开，但是显性边界符后的代码将被视为 cell 标题，不再是可执行的代码（图 1.3.1 行 11）。

```
1       %% 变量定义
2 -     clear;
3 -     clc;
4 -     num = 100;                  % 定义分子
5 -     den = [3,60,100];           % 定义分母
6 -     h = tf(num,den);            % 建立状态方程
7 -     s = ss(h);                  % 转化到状态空间
8       %% 计算响应
9 -     [yi,ti,xi]=impulse(s);      % 单位冲激响应
10 -    [ys,ts,xs]=step(s);         % 单位阶跃响应
11      %% 绘图 plot(ti,yi,'-k');
12 -    hold on;
13 -    plot(ts,ys,'--r');
```

图 1.3.1　显性边界

M 文件的开始与结尾视为隐性 cell 边界。对于 M 函数文件，函数声明行与对应的 end 指令也作为隐性边界。如图 1.3.2 所示的函数，由于插入显性 cell 边界符，function str = leapyear(x)内的代码被视为两个 cell。

如果 M 文件、M 函数体或控制流语句里没有显性边界符，整个文件或函数体将被视为一个 cell，这样的 cell 不会被高亮显示。另外由于存在隐性 cell 边界，用户可以不必明确定义第一个 cell 的起始位置，如图 1.3.2 所示。

```
1      function str = leapyear( y )
2      %% 判断一个年份是否为闰年
3 -      if rem(y,4)==0          % 年份能被4整除
4 -        if rem(y,100)==0      % 是否是整100年
5 -          if rem(y,400)==0    % 整100年能否被400整除
6 -              leap=1;         % 整400年，是闰年
7 -          else
8 -              leap=0;         % 整100年但非整400年，非闰年
9 -          end
10 -       else
11 -           leap=1;           % 被4整除但非整100年，是闰年
12 -       end
13 -     else
14 -         leap=0;             % 不能被4整除，非闰年
15 -     end
16     %% 输出结果
17 -     if leap==1
18 -        str = strcat([num2str(y),' IS a leap year.']); % 若leap为真，输出肯定字符串
19 -     else
20 -        str = strcat([num2str(y),' is NOT a leap year.']); % 若leap为真，输出否定字符串
21 -     end
22 -  end
```

图 1.3.2　隐性边界

2. 删除 cell

(1) 删除 cell 边界符所在的整行。

(2) 删除 cell 边界符其中的一个百分号。

(3) 删除 cell 边界符与 cell 标题之间的空格。

3. 使用 cell 调试模式

考察例 1.2.3 二阶网络脚本，可以将其划分为 3 个 cell：首先是变量定义，其次是计算响应，最后则是绘图代码。

(1) 查看 MATLAB 编辑器 Cell 菜单项下的第一个选项 Enable Cell Mode/Disable Cell Mode，确保 cell 调试模式已启用。

(2) 将光标依次定位在行 1"clear;"、行 6"[yi,ti,xi]=impulse(s);"、行 9"plot(ti,yi,'-k');"的代码之前，并单击编辑器窗口工具栏按钮，创建 3 个 cell，当前的 cell 将高亮显示，如图 1.3.3 所示。

(3) 对于大型的 M 文件，建议为每个 cell 添加标题，方便快速定位，单击工具栏按钮，显示各 cell 的标题，如图 1.3.4 所示。

(4) 选择 cell 工具栏"变量定义"命令，按下两次按钮，观察工作空间的变量值是否符合预期，在初步认定代码正确的情况下，再按按钮，最后执行 cell"绘图"命令。

这样就避免了在尚未明确代码是否正确时，重复执行绘图等代码，加快了 M 文件的调试。

(5) 按钮、、用于直接进入下一个或上一个 cell 以及执行整个 M 文件，默认情况下这 3 个按钮不显示在编辑窗口，用户可选择 cell 工具栏的右键菜单项 Customize...，在打开的选项窗口中启用这三个按钮，如图 1.3.5 所示。

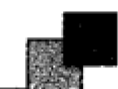




图 1.3.3　创建 cell



图 1.3.4　命名 cell

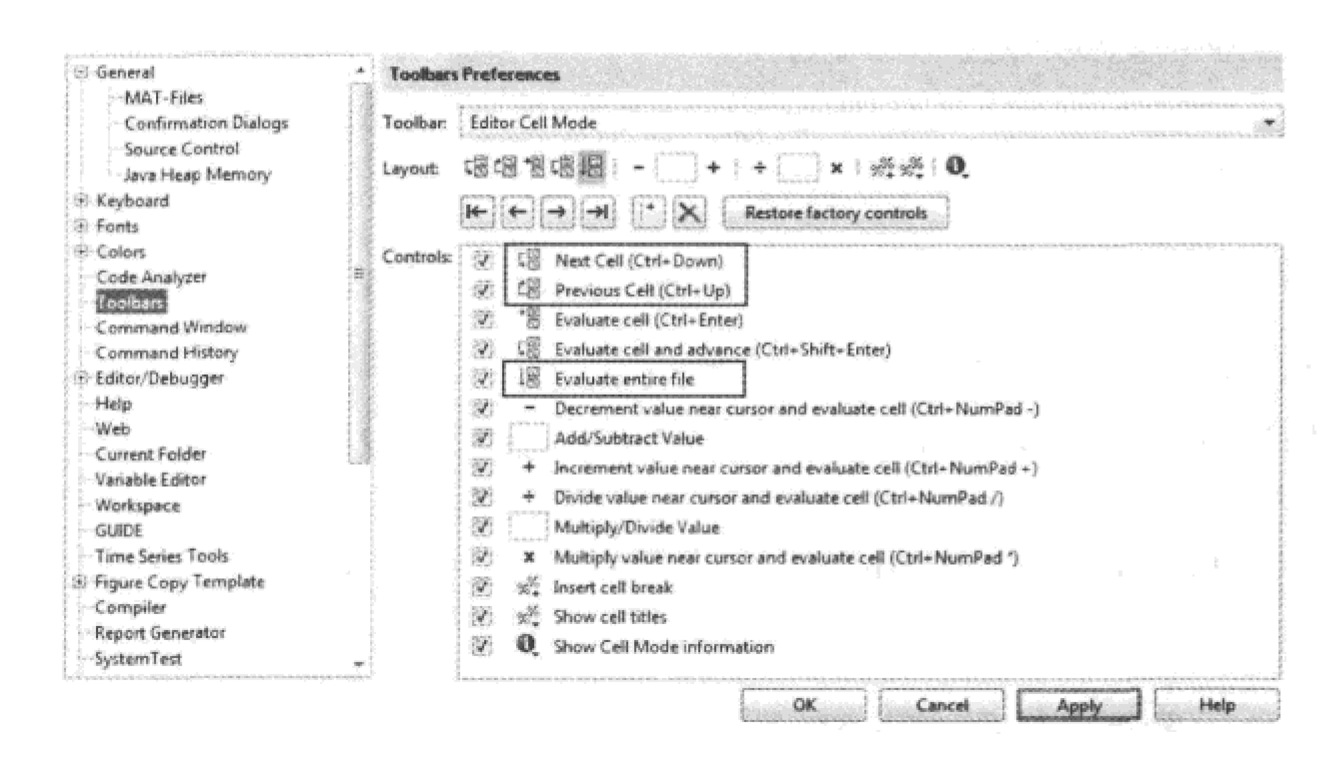


图 1.3.5　启用 cell 按钮

1.4　M 文件的发布

有些时候，用户需要在个人博客、会议、项目组上公开自己完成的 M 代码及结果。MATLAB 为此提供了代码发布工具，它将用户代码及其运行全过程以 HTML、LaTeX 、doc、PPT 等格式(R2009b 还新增了 PDF 格式)形成一份完整的文件，便于直接使用或二次编辑。

发布 M 代码的方法有两种：

(1) 在编辑窗口选择菜单项 File→Publish xxx 或按钮，使用默认配置发布 M 代码；

(2) 在编辑窗口选择菜单项 File → Publish Configuration for xxx. m → Edit Publish Configuration for xxx. m...(图 1.4.1)或按钮右侧的下拉箭头→ Edit Publish Configuration for xxx. m...，修改配置选项后再行发布，如图 1.4.2 所示。

方法(1)一般用于直接发布 M 脚本文件，而发布 M 函数文件通常需要定义输入变量，因此第一次发布 M 函数需要使用方法(2)，此后则可使用方法(1)再次发布。

以例 1.2.5 代码，说明发布的过程：



图 1.4.1　选择发布菜单项

(1) 打开 M 函数文件 ruku. m。

(2) 选择按钮右侧的下拉箭头→ Edit Publish Configuration for ruku. m. . . 。

(3) 在 MATLAB expression 文本框中输入函数调用代码,并按需要修改发布文档的格式以及所包含的内容,如图 1.4.2 所示。

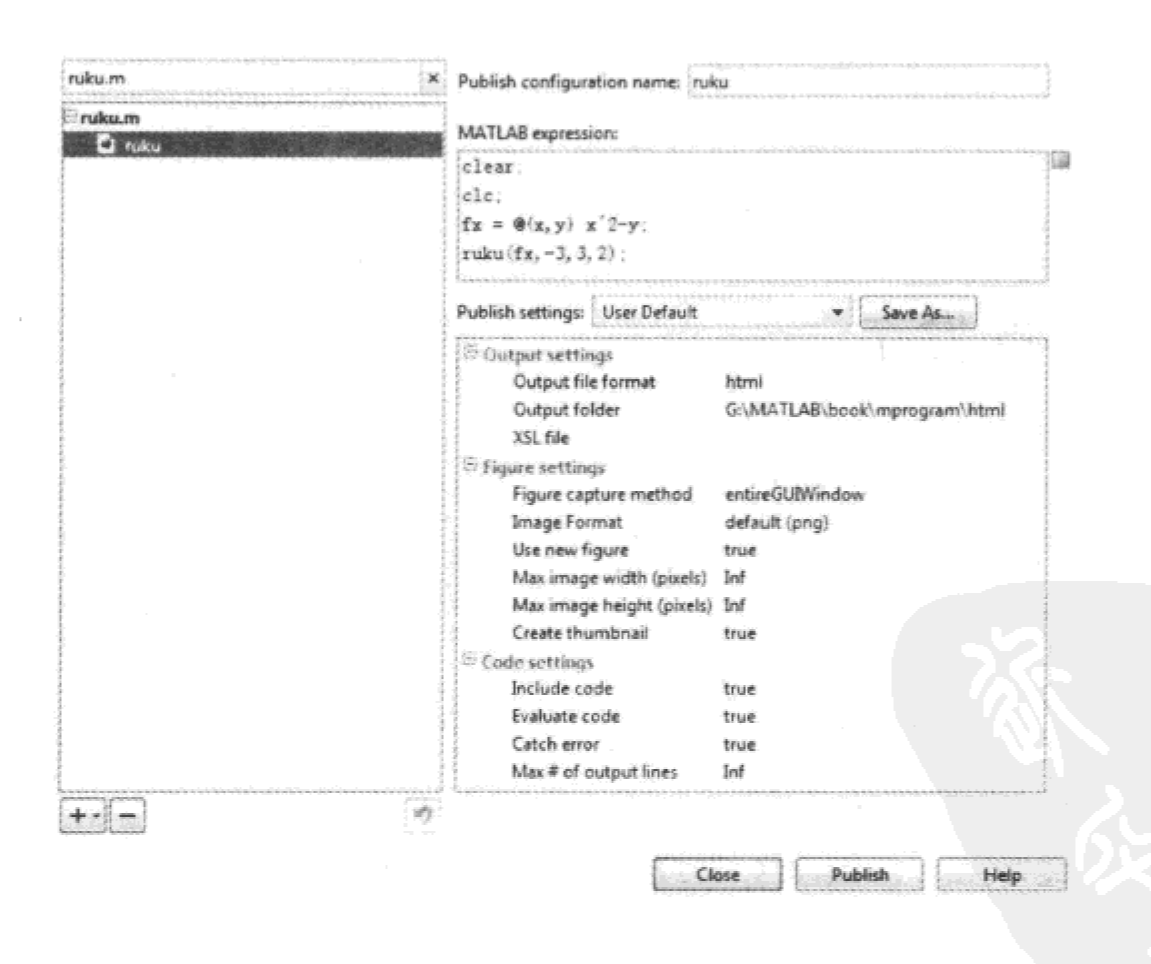
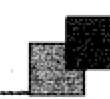


图 1.4.2　发布配置选项

(4) 生成报告文档,如图 1.4.3 所示。



Contents


```
function y = ruku(fx, xmin, xmax, y0)

% 采用四阶龙格库塔法数值积分
% fx为函数名
% x0为自变量的初始值
% y0为因变量的初始值
h = 0.01;              % 步长
x = xmin:h:xmax;           % 自变量
n = length(x);         % 计算点数
y = zeros(1,n);        % 初始化y向量
y(1) = y0;             % y初值
```

四阶龙格库塔法

```
for i=1:n
```

k1

```
    k1 = feval(fx,x(i),y(i));
```

```
end
```

绘图

```
y = y(1:n);
plot(x,y);
title('4 order Runge-Kutta');
xlabel('x');            % X轴标签
ylabel('y');            % Y轴标签
grid on;
```



```
end
```

图 1.4.3　生成报告

1.5 Embedded MATLAB

Embedded MATLAB 是 MATLAB 的一个子集，它支持从概念到实现的编程理念。其优势在于：可用易于编程的 MATLAB 语言替代 C 语言；用 MATLAB 语言编写的 M 代码可直接生成高效的嵌入式 C 代码或 HDL 代码；特别是 R2010b 支持可变大小数据的动态内存分配，大大提高编程效率，降低编程难度。读者在后续章节将看到 Embedded MATLAB 非凡的表现力。

作者在《基于模型的设计及其嵌入式实现》一书中详细介绍了 Embedded MATLAB 的编程规范以及注意事项，读者可以阅读相关部分，而更全面、更权威的 Embedded MATLAB 信息请参考 MathWorks 公司的技术手册。本节以一个实例简要介绍 Embedded MATLAB 的应用。

1.5.1 Embedded MATLAB 的主要功能特点

1. Embedded MATLAB 支持的功能(R2010b 版)

(1) 支持 400 多个 MATLAB 运算符、浮点函数，超过 120 个定点工具箱函数，以及超过 70 个信号处理函数。

(2) 支持多维阵列、实数和复数、结构、流程控制和下标运算等高级 MATLAB 语言功能。

(3) 支持直接在 M-code 文件中调用 Real-Time Workshop 工具箱。

(4) 支持在 Simulink 中嵌入 Embedded MATLAB 用户自定义模块。

(5) 支持在 Stateflow 中嵌入 Embedded MATLAB 函数模块。

(6) 支持加快 M-code 文件定点算法的仿真速度。

(7) 支持直接从 M-code 文件生产 HDL 代码。

(8) 支持集成已存在的用户 C 语言算法到 Embedded MATLAB 模块中。

(9) 支持可变大小数据(R2009b 新增)。

(10) 支持全局变量(R2010a 新增)。

(11) 支持结构体数据类型(R2010a 新增)。

(12) 支持可变大小数据的动态内存分配(R2010b 新增)。

(13) 支持增强编辑报告(R2010b 新增)。

(14) 支持用更少的静态内存来生成 MEX 函数(R2010b 新增)。

2. Embedded MATLAB 所受到的限制

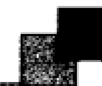
(1) 不支持元胞数组。

(2) 不支持 MATLAB 对象。

(3) 不支持内嵌函数。

(4) 不支持稀疏矩阵。

(5) 不支持可视化。



(6) 不支持 Java。

(7) 不支持 try/catch 语句。

(8) 不支持 Matrix deletion。

(9) 不支持枚举数据的冒号操作(R2010b 新增)。

Embedded MATLAB 子集如图 1.5.1 所示,具体 Embedded MATLAB 函数列表见附录。

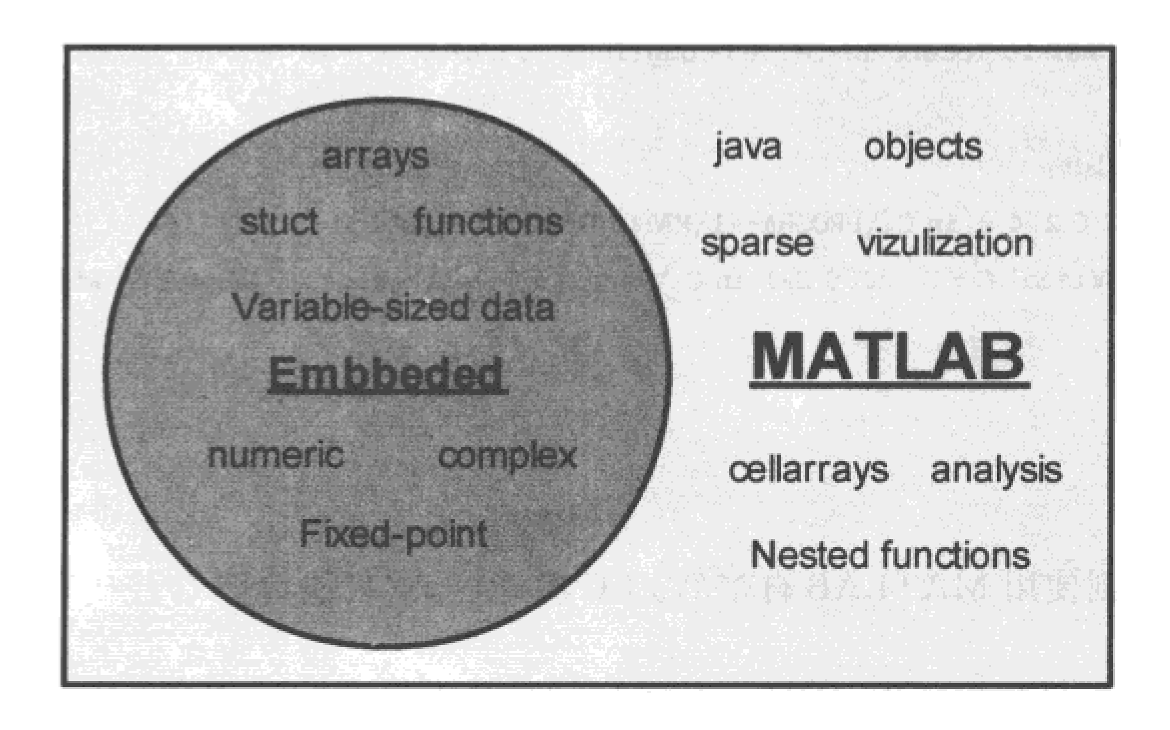


图 1.5.1 Embedded MATLAB 子集

1.5.2 Embedded MATLAB 的编程规范

1. Embedded MATLAB 遵守的规范

(1) 首先必须满足 MATLAB 程序的语法规范。

(2) 其次还得满足 Embedded MATLAB 语法规范。

2. Embedded MATLAB 规范检查的步骤

(1) 首先用 cell 将程序分段,然后用 M-Lint 对代码进行 MATLAB 程序语法检查,并根据提示修改错误,缩小错误范围。

(2) 在 M 文件开头或第一行末尾加上 Embedded MATLAB 编辑指令%# eml,继续用 M-Lint 对 M-code 进行设计时的 Embedded MATLAB 语法检查,并根据 M-Lint 的建议修改错误。

(3) 用 emlmex 指令对 M 代码进行 C 代码生成时的 Embedded MATLAB 兼容性检查,并根据错误报告修改错误。

1.5.3 C 编译器的设置

在实现 MATLAB 编译器的各种功能之前(如 mex),需要指定 MATLAB 编译器。

C 编译器的设置过程如下:

```
>> mex - setup
Please choose your compiler for building external interface (MEX) files:

Would you like mex to locate installed compilers [y]/n?
```

(1) 选择 y,按 Enter 键,代码如下:

```
Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc - win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2010b\sys\lcc
[2] Microsoft Visual C + + 2008 SP1 in C:\Program Files\Microsoft Visual Studio 9.0

[0] None

Compiler:
```

(2) 选择 1,则使用 MATLAB 自带的 LCC 编译器,它只能编译 C 代码,不能编译 C++ 代码。代码如下:

```
Compiler: 1

Please verify your choices:

Compiler: Lcc - win32 C 2.4.1
Location: C:\PROGRA~1\MATLAB\R2010b\sys\lcc
```

(3) 选择 2,则使用作者机器上安装的 Microsoft Visual C++ 2008 作为 C 编译器。代码如下:

```
Compiler: 2

Please verify your choices:

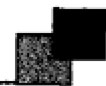
Compiler: Microsoft Visual C + + 2008 SP1
Location: C:\Program Files\Microsoft Visual Studio 9.0

Are these correct [y]/n?
```

(4) 选择 y,按 Enter 键完成 C 编译器的设置,代码如下:

```
Are these correct [y]/n? y

***************************************************************************
  Warning:MEX - files generated using Microsoft Visual C + + 2008 require
          that Microsoft Visual Studio 2008 run - time libraries be
```



```
          available on the computer they are run on.
          If you plan to redistribute your MEX-files to other MATLAB
          users, be sure that they have the run-time libraries.
***************************************************************************

Trying to update options file: C:\Documents and Settings\NO.1\Application Data\MathWorks\MAT-
LAB\R2010b\mexopts.bat
From template: C:\PROGRA~1\MATLAB\R2010b\bin\win32\mexopts\msvc90opts.bat

Done . . .

***************************************************************************
  Warning: The MATLAB C and Fortran API has changed to support MATLAB
          variables with more than 2^32-1 elements. In the near future
          you will be required to update your code to utilize the new
          API. You can find more information about this at:
          http://www.mathworks.com/support/solutions/en/data/1-5C27B9/?solution=1-5C27B9
          Building with the -largeArrayDims option enables the new API.
***************************************************************************
```

1.5.4 Embedded MATLAB 编程实例

沿用例 1.2.4 来说明 Embedded MATLAB 程序的编程规范、调试过程以及嵌入式 C 代码生成过程。

1. MATLAB 语法检查

为了说明 MATLAB 语法检查的过程，以下代码引入了若干错误：

```
function str = leapyear( y )
%% 判断一个年份是否为闰年
  if rem(y,4) = 0               % 年份能被 4 整除
   if rem(y,100)== 0            % 是否是整 100 年
    if rem(y,400)== 0           % 整 100 年能否被 400 整除
      leap = 1;                 % 整 400 年,是闰年
    else
      leap = 0;                 % 整 100 年但非整 400 年,非闰年
    end
   else
     leap = 1;                  % 被 4 整除但非整 100 年,是闰年
   end
  else
```

```
    leap = 0;                % 不能被 4 整除,非闰年
  end

%% 输出结果
  if leap == 1
    str = strcat([num2str(y),'IS a leap year.']);
                             % 若 leap 为真,输出肯定字符串
  else leap == 0
    str = strcat([num2str(y),'is NOT a leap year.');
                             % 若 leap 为真,输出否定字符串
  end

end
```

编辑器窗口右侧 M-Lint 消息区出现多个带有颜色的横条,上方的小方块显示当前 M 文件的语法检测状态,如图 1.5.2 所示。

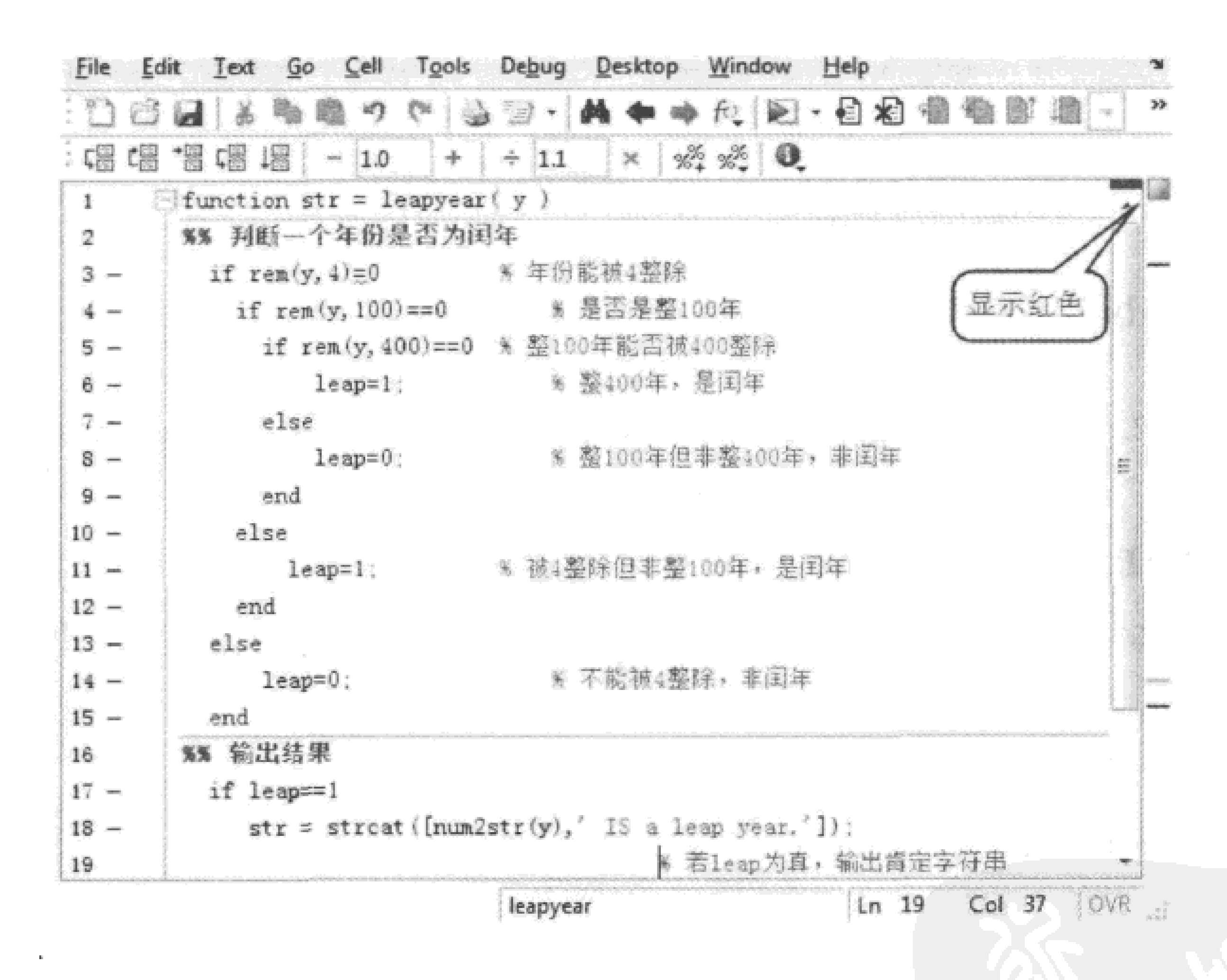


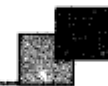
图 1.5.2　MATLAB 语法检查结果

(1) 红色：M 文件至少有一个语法错误。对于某些错误,M-Lint 会高亮显示,如未结束的字符串、不匹配的关键词、圆括弧、花括弧、方括弧等。

(2) 橘黄色:M 文件有警告或者值得改进的代码,但没有语法错误。

(3) 绿色：M 文件没有任何语法错误、警告以及可改进的代码。

同样地,代码下方带有红色波浪线的表示有语法错误,代码下方带有橘黄色波浪线的表示有警告或待改进。首次单击小方块,光标将定位至第一个包含 M-Lint 消息的代码段,再次单



击则定位至下一个代码段；当然用户也可以直接将鼠标指针停留在 M-Lint 消息区的颜色横条或带有波浪线的代码段，M-Lint 即给出提示或修改意见。

如图 1.5.3 所示，M-Lint 提示第 3 行存在语法错误。

检查第 3 行代码可以发现，if 的判断条件应使用“==”，而不是“＝”，按此修改后，代码下方的波浪线与消息区的颜色横条即消失了。

```
1      function str = leapyear( y )
2        %% 判断一个年份是否为闰年
3 -        if rem(y, 4)=0      Line 3: Parse error at '=': usage might be invalid MATLAB syntax.
4 -          if rem(y, 100)==0        % 是否是整100年
5 -            if rem(y, 400)==0  % 整100年能否被400整除
6 -              leap=1;              % 整400年，是闰年
```

图 1.5.3　第 3 行错误提示

针对第 20 行，M-Lint 给出了提示 1：Line 20：Possible inappropriate use of == operator. Use ＝ if assignment is intended 与提示 2：Line 20：Terminate statement with semicolon to suppess output (in functions).，并各提供了一个 Fix 按钮。如果按下第 1 个按钮，系统自动用＝替换==，按下第 2 个按钮，系统自动在行末加入一个分号，如图 1.5.4 所示。另外单击提示 2 的蓝色链接，还可以查看详细的修改意见，如图 1.5.5 所示。

但详细分析代码，if 条件语句的 else 分支无须再作判断，而 M-Lint 将 leap == 0 当做普通的赋值语句，因此显然应直接删除该代码。

```
17 -     if leap==1
18 -   Line 20: Possible inappropriate use of == operator. Use = if assignment is intended.  Fix
19     Line 20: Terminate statement with semicolon to suppress output (in functions).      Fix
20 -     else leap==0
21          str = strcat([num2str(y),' is NOT a leap year.']);
22                                     % 若leap为真，输出否定字符串
```

图 1.5.4　第 20 行错误提示

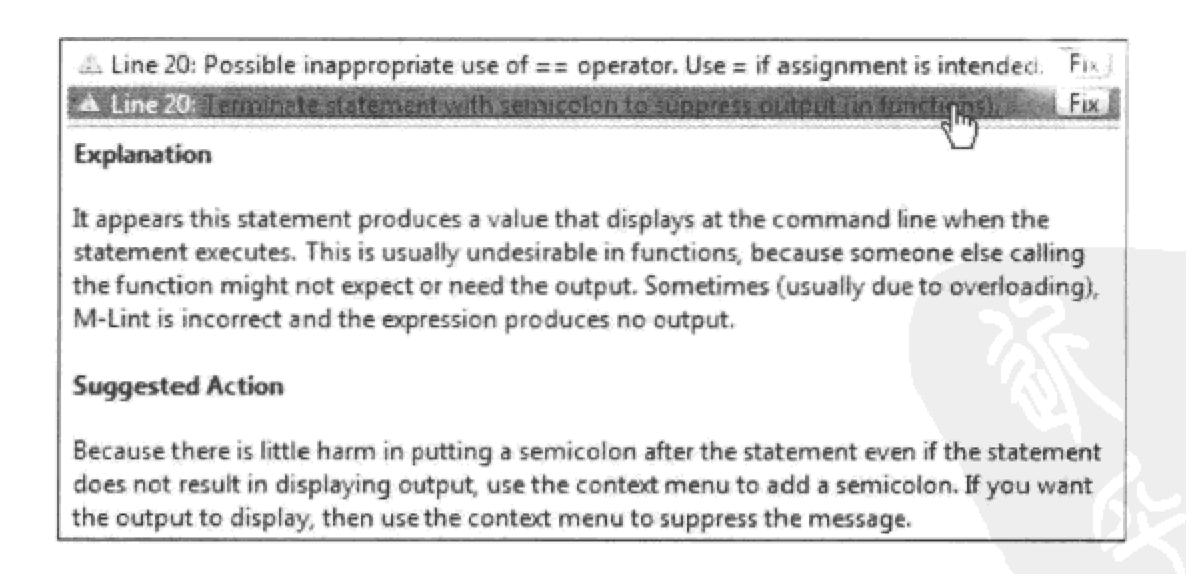


图 1.5.5　修改意见

第 20 行的修改方式说明，M-Lint 并不能针对各种情况给出完全正确的判断，有些时候用户不需要修改代码，也不需要 M-Lint 给出提示信息，因而要屏蔽 M-Lint 消息，当然，用户不

能屏蔽 M-Lint 的语法错误提示。

屏蔽消息有多种实现方法：

(1) 针对当前文件的某种情况，屏蔽一次 M-Lint 消息。

(2) 针对当前文件的某种情况，均屏蔽 M-Lint 消息。

(3) 针对所有文件的各种情况，均屏蔽 M-Lint 消息。

(4) 设置默认的 M-Lint 消息规则。

(5) 设置用户自定义的 M-Lint 消息规则。

右击带有波浪线的代码段，查看菜单项 Suppress “Possible inappropriate use of ==... ”下的选项，如图 1.5.6 所示。

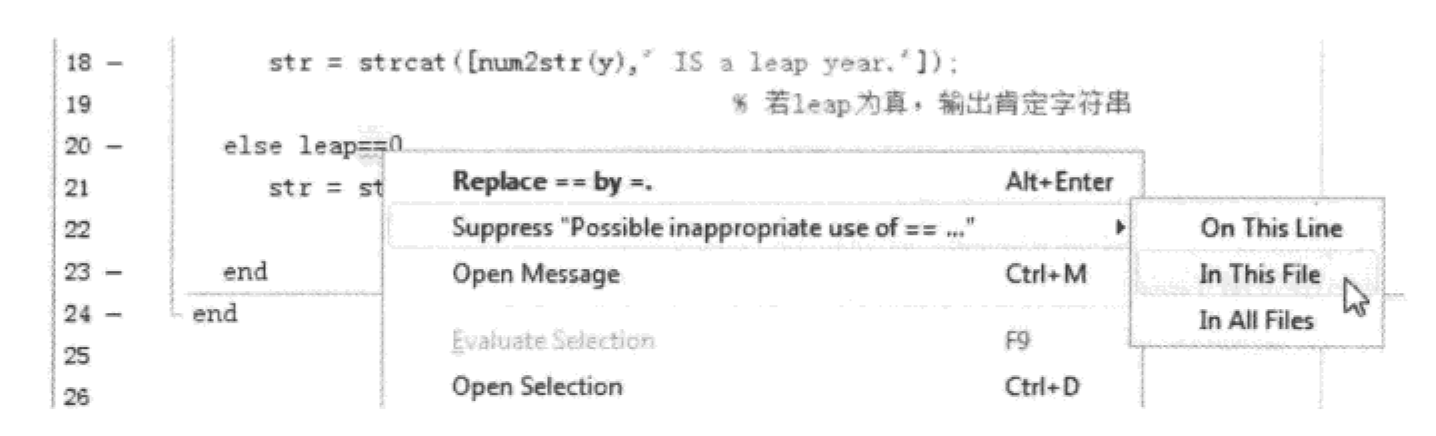


图 1.5.6　屏蔽 M-Lint 消息菜单

(1) On This Line 表示针对当前行，屏蔽一次 M-Lint 消息。

(2) In This File 表示针对当前文件，均屏蔽 M-Lint 消息。

(3) In All Files 表示所有文件，均屏蔽 M-Lint 消息。

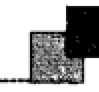
由于该行存在两个 M-Lint 提示，用户需要进行两次屏蔽消息的操作，最后 M-Lint 将在代码末尾加入%#ok<*NOPRT,*EQEFF>，表示在当前文件不再对该类代码提出警告，如图 1.5.7 所示。

```
16        %% 输出结果
17 -        if leap==1
18 -            str = strcat([num2str(y),' IS a leap year.']);
19                                          % 若leap为真，输出肯定字符串
20 -        else leap==0   %#ok<*NOPRT,*EQEFF>
21              str = strcat([num2str(y),' is NOT a leap year.');
22                                          % 若leap为真，输出否定字符串
23 -        end
```

图 1.5.7　屏蔽消息的效果

最后一个 M-Lint 消息提示第 21 行存在语法错误，可能是缺少某一类的括弧，如图1.5.8 所示。对照检查第 19 行代码可以发现，代码中缺少一个“]”。

根据提示，完成了所有代码修改，M-Lint 消息区的小方块显示绿色，表示代码通过了 MATLAB 的代码规范检查，如图 1.5.9 所示。



```
17 -      if leap==1
18 -         str = strcat([num2str(y),' IS a leap year.']);
19                          Line 21: Invalid syntax at ')'. Possibly, a ), }, or ] is missing.
20 -      else
21           str = strcat([num2str(y),' is NOT a leap year.'});
22                                    % 若leap为真，输出否定字符串
23 -      end
```

图 1.5.8　第 21 行错误提示

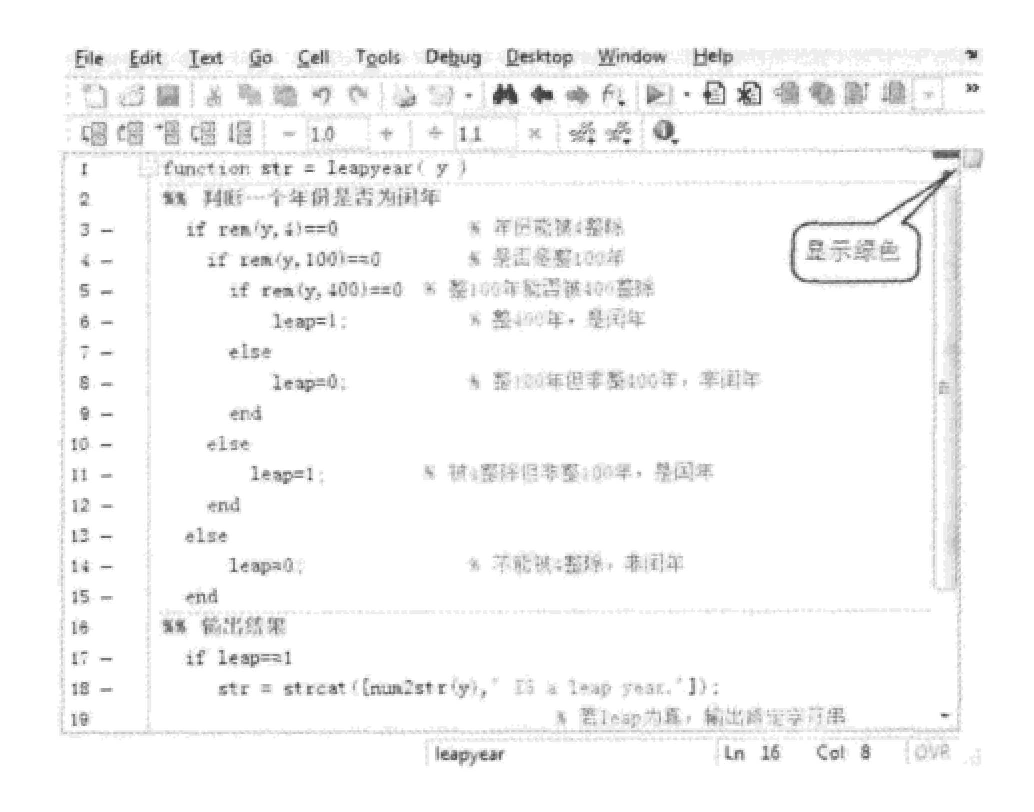


图 1.5.9　通过了代码规范检查

2. 设计时 Embedded MATLAB 语法检查

在程序的第一行末尾或下一行加上指令%#eml，M-lint 消息区的小方块颜色没发生变化，表示通过设计时规范语法检查，如图 1.5.10 所示。

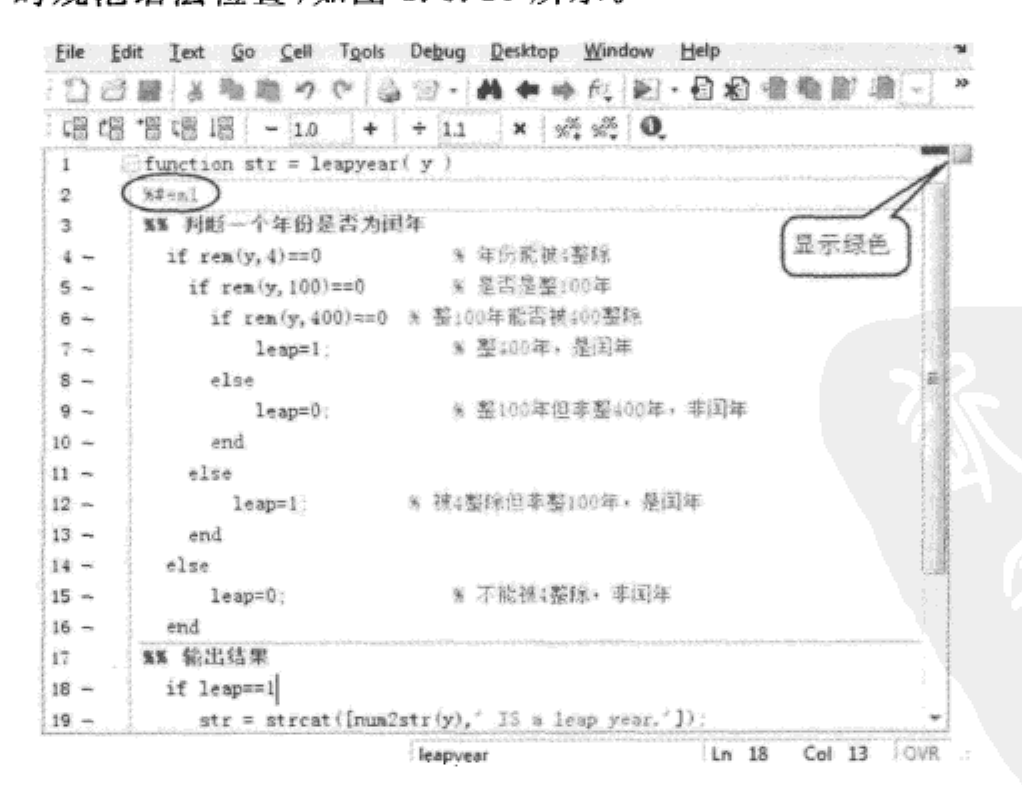


图 1.5.10　通过设计时规范检查

3. 代码生成时 Embedded MATLAB 兼容性检查

在命令行输入命令：

```
>> emlmex leapyear - report
```

命令行继而显示以下错误信息，说明检查失败：

```
??? The function'num2str' is not supported by Embedded MATLAB for code generation. See the documen-
tation for eml.extrinsic to learn how you can use this function in simulation.
Error in ==> leapyear Line: 19 Column: 20
C - MEX generation failed: Open error report.
??? Error using ==> emlmex
```

单击 Open error report 按钮打开错误报告，如图 1.5.11 所示，提示信息指出 Embedded matlab 无法生成 num2str 等函数的代码，为此需要对代码进行改造。

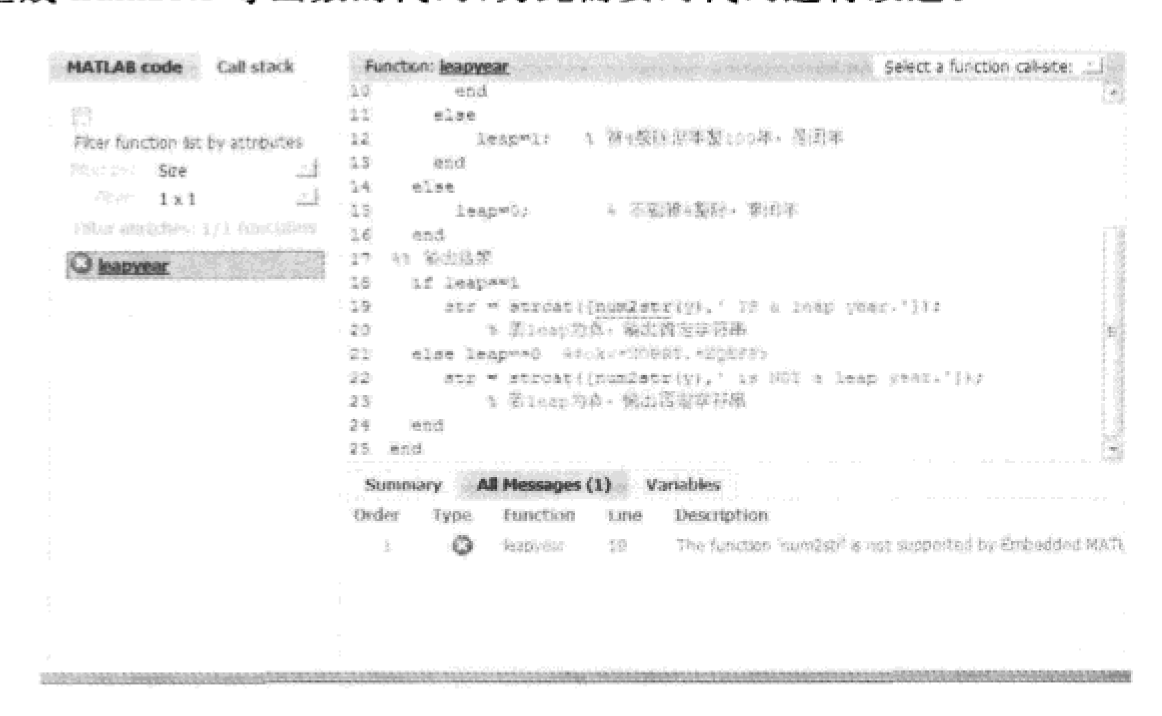


图 1.5.11 错误报告

将代码的输出部分修改如下：

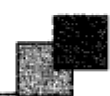
```
...
 if leap == 1
   str = 1;
 else
   str = 0;
 end
...
```

在命令行继续输入命令：

```
>> emlmex leapyear - report
```

命令行显示以下错误信息，说明检查仍旧失败

```
??? Build error: Compilation returned error status code 2. See the target build log for further de-
tails.
Error in ==> leapyear Line: 1 Column: 1
```



```
C-MEX generation failed: Open error report.
??? Error using ==> emlmex
```

这是由于代码注释中包含了中文字符，将所有注释删除或用英文代替，再次创建C-MEX函数即可成功，如图1.5.12所示，生成报告如图1.5.13所示。

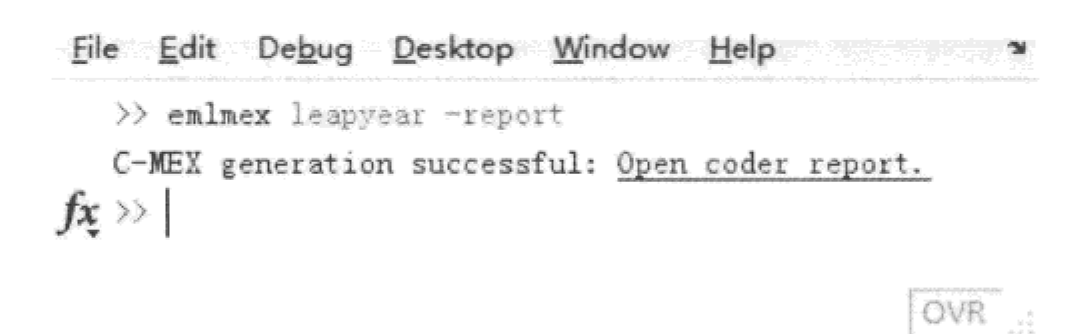


图1.5.12 创建C-MEX函数成功

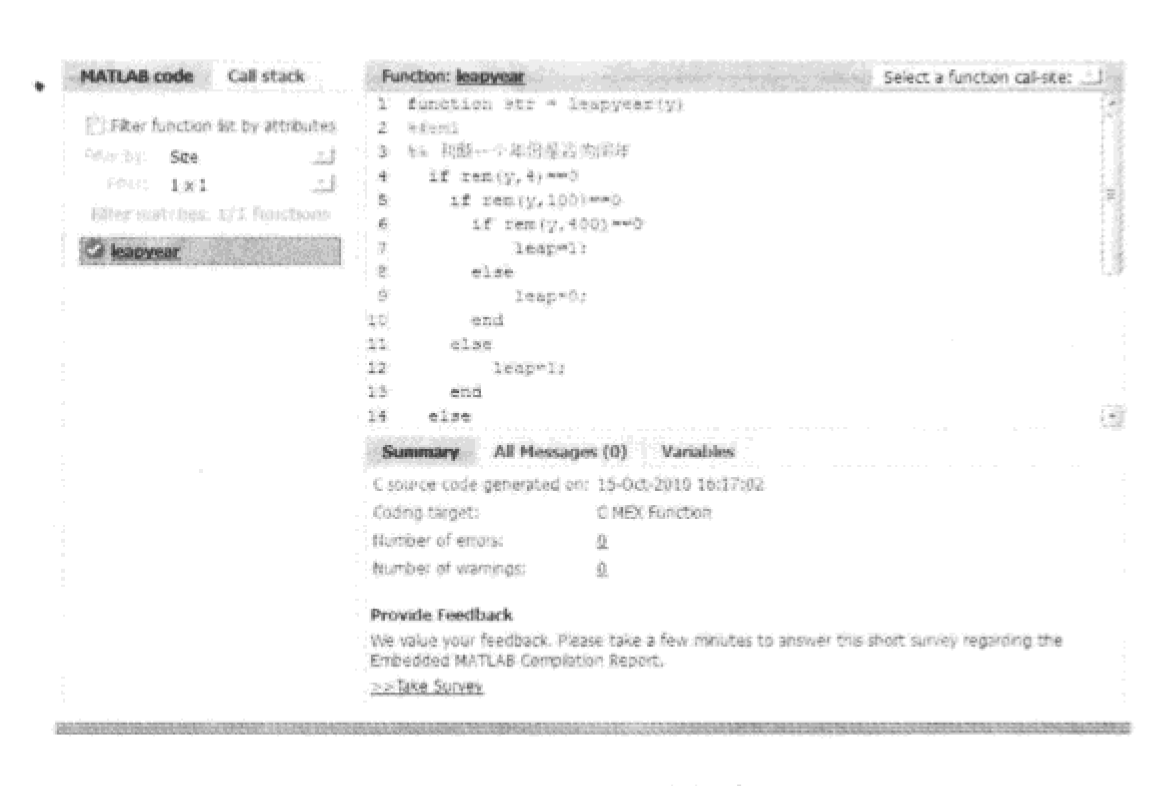


图1.5.13 成功报告

4. 生成嵌入式C代码

在命令行中输入命令，提示代码生成成功，如图1.5.14所示。

```
>> emlc leapyear -c -report -T rtw:lib
```

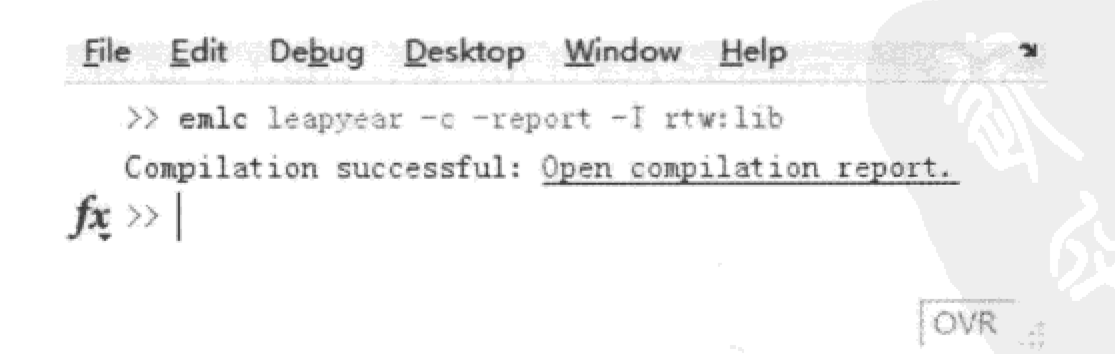


图1.5.14 生成嵌入式C代码成功

单击链接，查看生成报告，如图1.5.15所示。继续单击报告左侧导航窗口的C code页，查看生成的C代码，如图1.5.16所示。

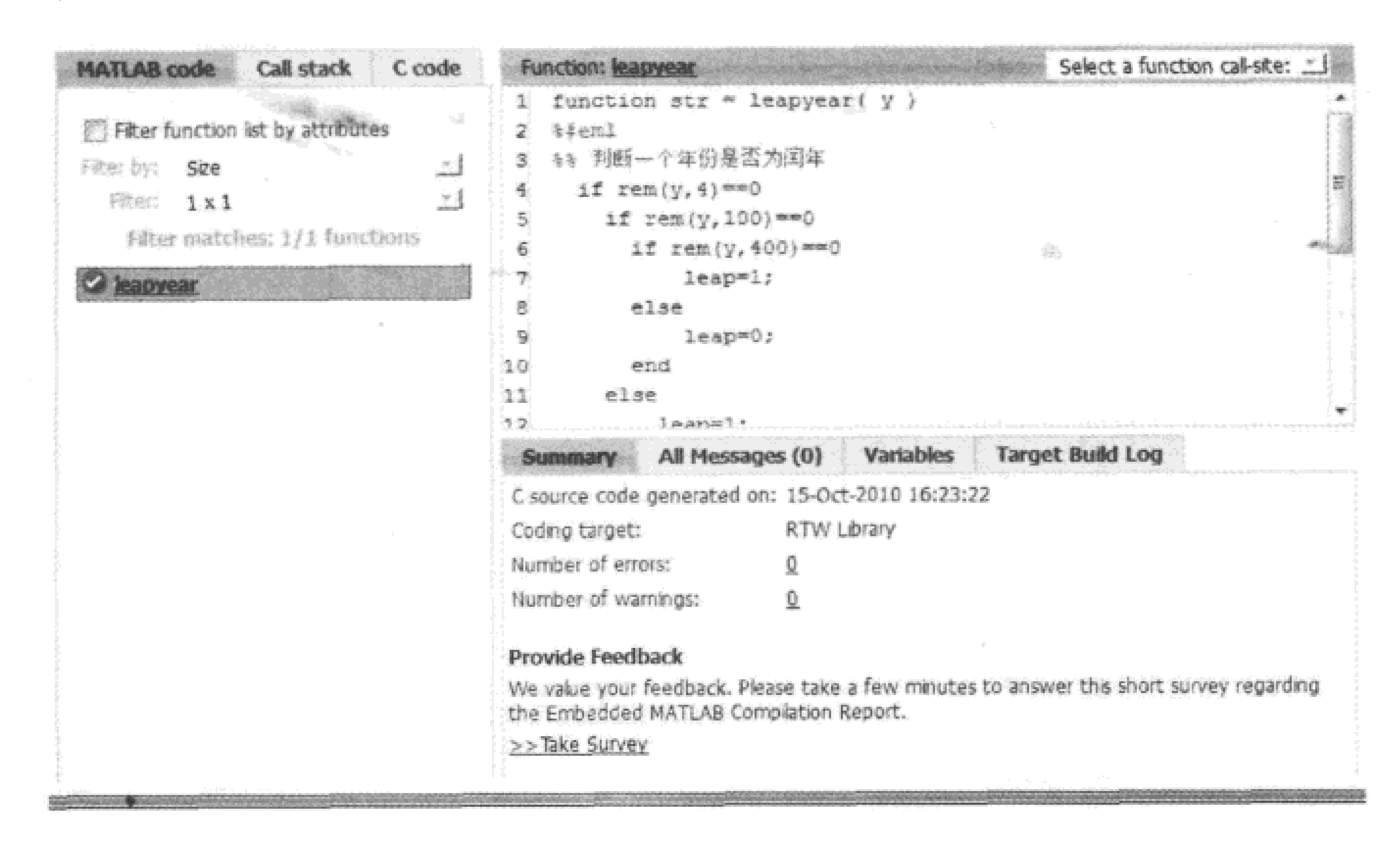


图 1.5.15　嵌入式 C 代码生成报告



图 1.5.16　生成的 C 代码

5. 加入 Embedded MATLAB 模块

在 Simulink 模块库找到 Embedded MATLAB 模块，如图 1.5.17 所示，添加到一个空白的模型，按图 1.5.18 连接，并将模型与 M 函数 leapyear.m 保存在同一个目录。

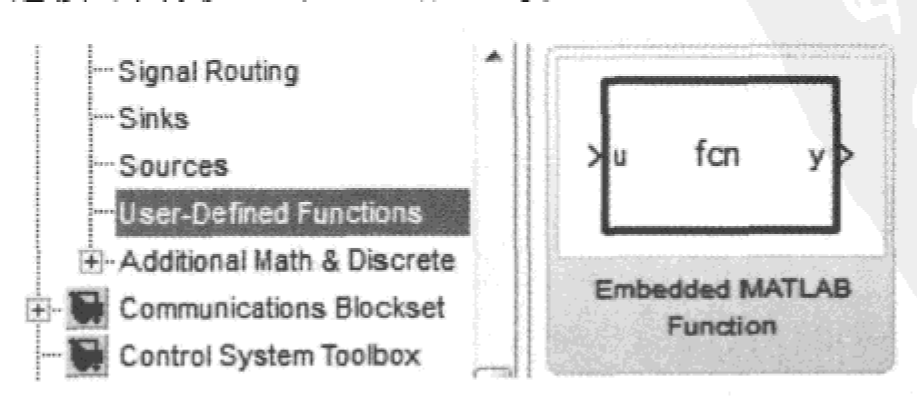


图 1.5.17　Embedded MATLAB 模块



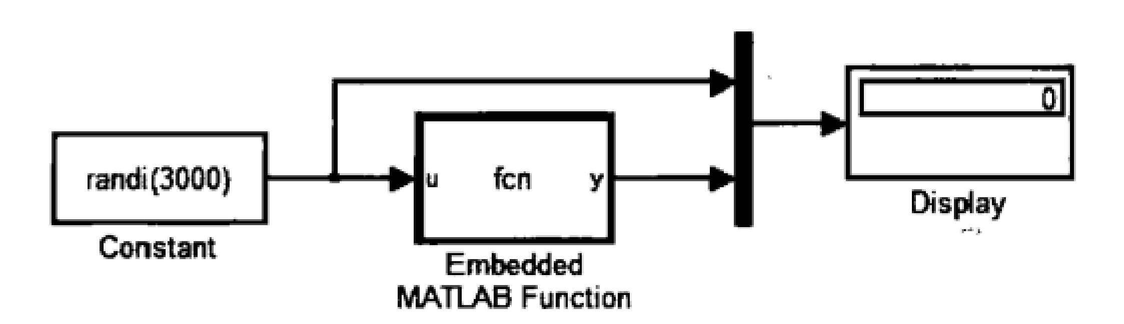


图 1.5.18 连接模型

双击打开 Embedded MATLAB 模块,在编辑窗口加入以下代码,即可调用该函数。

```
function y = fcn(u)
% #eml
y = leapyear(u);
```

仿真结果如图 1.5.19 所示,显示模块第二行的"1"说明 2912 年是闰年。

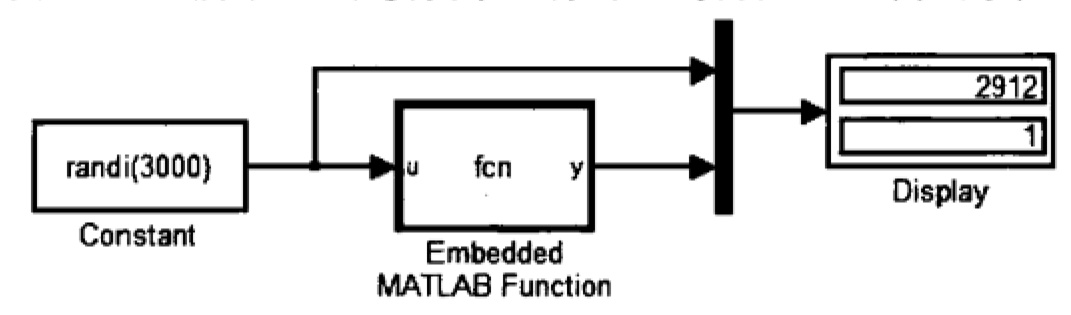


图 1.5.19 仿真结果

第2章

Simulink建模与调试

Simulink是动态和嵌入式等系统的建模与仿真工具,也是基于模型设计的基础。对于机电、航空航天、信号处理、自动控制、通信、音视频处理等众多领域,Simulink提供了交互式的可视化开发环境和可定制的模块库,对系统进行建模、仿真与调试等。并可实现与Stateflow有限状态机的无缝连接,扩展对复杂系统的建模能力。

通过Simulink模块库自带的1000多个预定义模块,基本上可快速地创建基于MCU器件应用的系统模型。运用层次化建模、数据管理,子系统定制等手段,即使是复杂的嵌入式MCU应用系统,也能轻松完成简明精确的模型描述。

大量使用Embedded MATLAB来创建用户自己的算法模块,可大大加快建模速度。读者在后面的内容中,会经常看到用Embedded MATLAB创建的算法模块,加快MCU器件开发的实例。模型是基于模型设计的起点,同时也是最核心的东西。本章将以基于PID控制的直流电动机的物理建模与调试为例来介绍Simulink,更详细的内容请读者参考MathWorks公司相关内容的用户手册。

Simulink的主要特点如下:

(1) 众多可扩展的模块库。

(2) 利用图形编辑器来组合和管理模块图。

(3) 以系统功能来划分模型,实现对复杂系统的管理。

(4) 利用模型浏览器(Model Explorer)寻找、创建、配置模型组件的参数与属性。

(5) 利用API实现与其他仿真程序的连接或集成用户代码。

(6) 用图形化的调试器和剖析器来检查仿真结果,评估模型的性能指标。

(7) 在MATLAB命令窗口中,可对仿真结果进行分析与可视化,自定义模型环境、信号参数和测试数据。

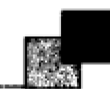
(8) 利用模型分析和诊断工具来确保模型的一致性,定位模型中的错误。

本章主要内容如下:

(1) Simulink基本操作。

(2) 搭建直流电动机模型。

(3) Simulink模型调试。



2.1　Simulink 基本操作

2.1.1　模块库和编辑窗口

1. 打开模型库浏览器

在 matlab 的命令窗口中输入 simulink 指令或单击 matlab 工具栏上的 simulink 图标就可以打开模型库浏览器，如图 2.1.1 所示。

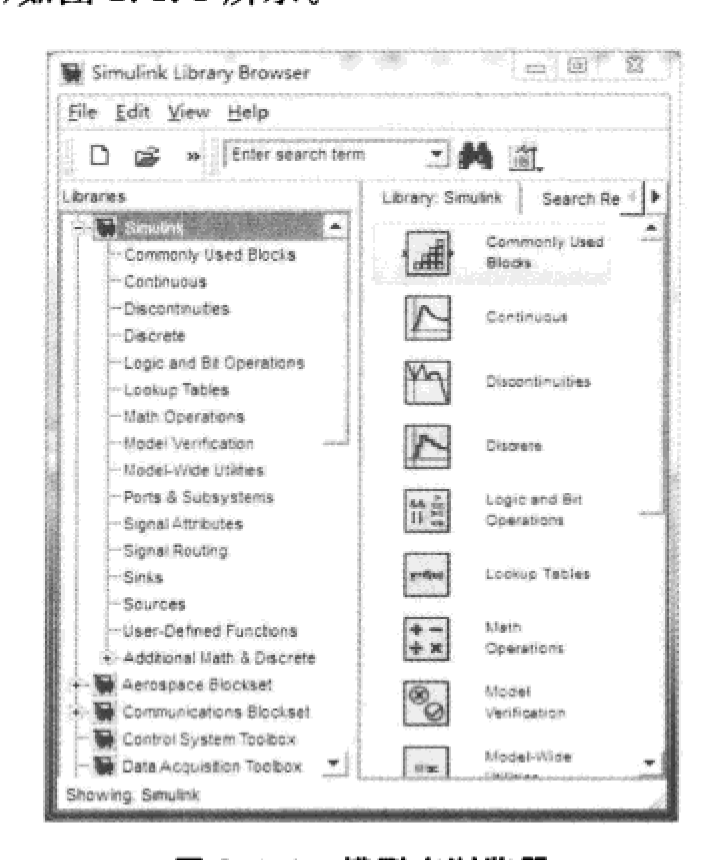


图 2.1.1　模型库浏览器

2. 打开模型编辑窗口

要建立一个新的模型，首先要打开一个模型编辑窗口。可以通过单击模块库浏览器上的 NEW Model 按钮，或 File→NEW→Model 来打开窗口，如图 2.1.2 所示。

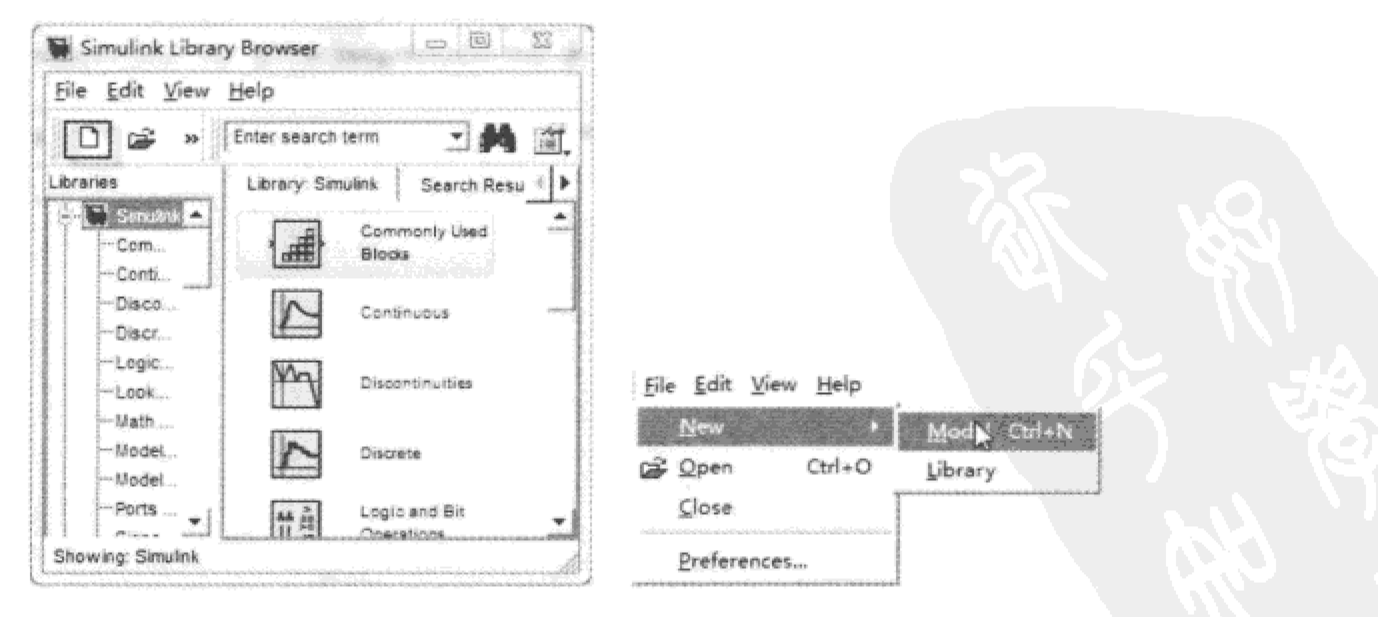


图 2.1.2　打开模型编辑窗口

2.1.2 Simulink 模块库

Simulink 模块库是建立模型的基础，其中囊括了大量的基本功能模块，只有当用户熟练地掌握了模块库，才能快速、高效地建立模型。从图 2.1.1 所示的模型库浏览器可知，在 Simulink 模块库中包含以下子模块库，如表 2.1.1 所列。

表 2.1.1 模块库列表

常用模块(commonly used block)	连续模块(continuous)
非连续模块(discontinuous)	离散模块(discrete)
逻辑和位操作模块(logic and bit operations)	查找表模块(lookup tables)
数学运算模块(math operations)	模型验证模块(model verification)
模型实用模块(model－wide utilities)	端口与子系统模块(ports & subsystems)
信号属性模块(signal attributes)	信号路由模块(signal routing)
接收器模块(sinks)	源模块(sources)
用户自定义模块(user－defined functions)	附加操作模块(additional math & discrete)

下面将详细介绍几种使用频率较高的模块库。

1. 常用模块库(commonly used block)

常用模块库中的模块是 simulink 所有模块库中使用频率最高模块的合集，主要是为了方便用户以最快的速度建立模型。常用模块包含图 2.1.3 所示的成员，模块功能如表 2.1.2 所列。

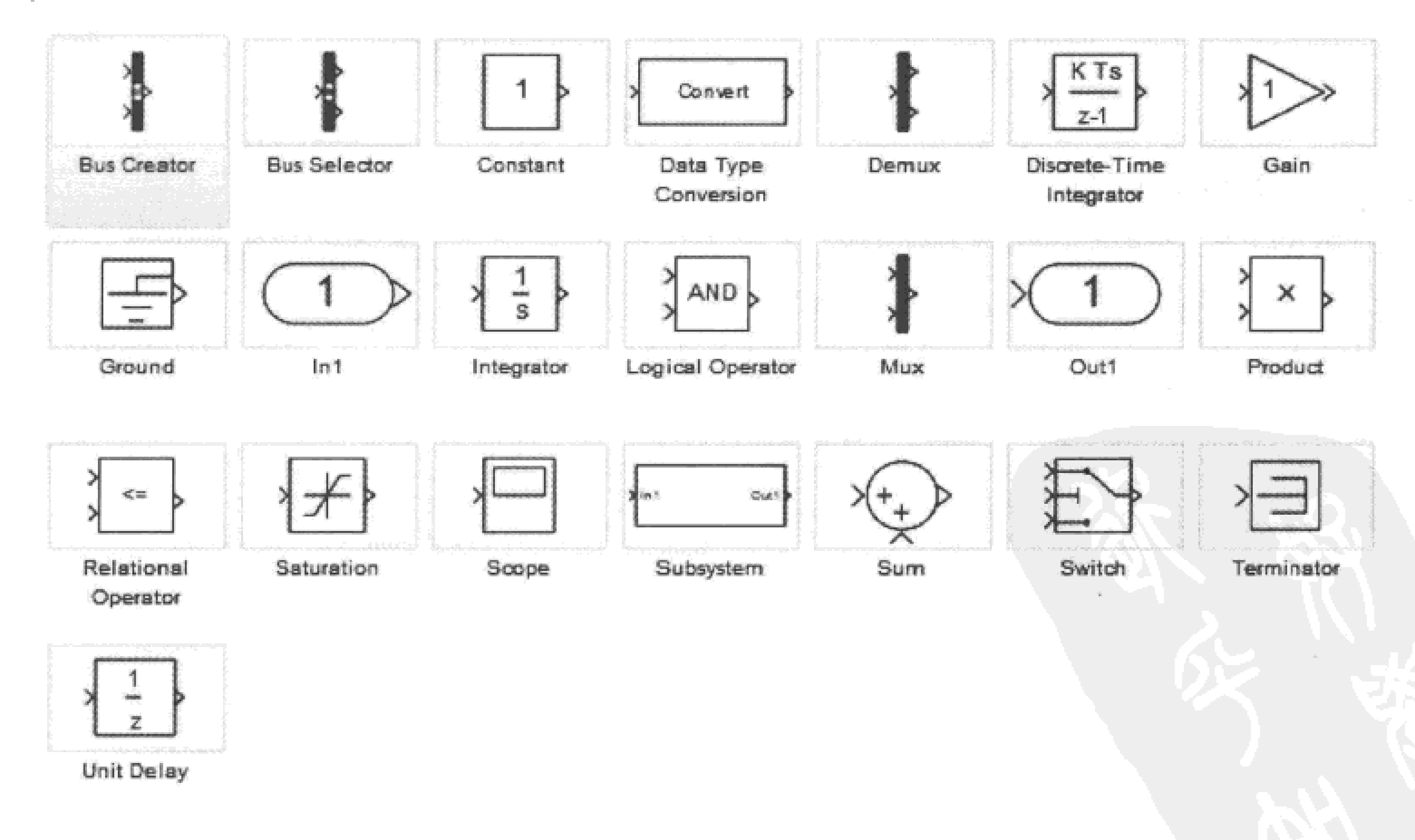


图 2.1.3 常用模块库

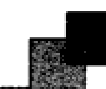


表 2.1.2 常用模块库列表

名 称	功 能	名 称	功 能
Bus Creator	生成总线	Bus Selector	分离总线
Constant	常量信号	Data Type Conversion	转换数据类型
Demux	抽取向量信号中的元素并输出	Discrete-Time Integrator	时间离散积分
Gain	放大器	Ground	接地
Inport	产生输入口	Integrator, Integrator Limited	信号积分
Logical Operator	逻辑运算	Mux	将输入信号合成为向量
Outport	产生输出口	Product	标量和非标量乘除或矩阵乘法和转置
Realational Operator	对输入作关系运算	Saturation	饱和
Scope and Floating Scope	显示仿真信号	Subsystem, Atomic Subsystem, Nonvitual Subsystem, CodeReuse Subsystem	以子系统表示其他系统
Sum, Add, Subtract, Sum ofElements	加或减	Switch	通过第二个输入值来输出第一或第二个输入
Terminator	终止未连接的输出口	Unit Delay	延迟一个采样周期

2. 连续模块库(continuous)

连续模块库中的模块如图 2.1.4 所示,它包含了搭建连续系统所涉及的绝大部分模块,这些模块的功能如表 2.1.3 所列。

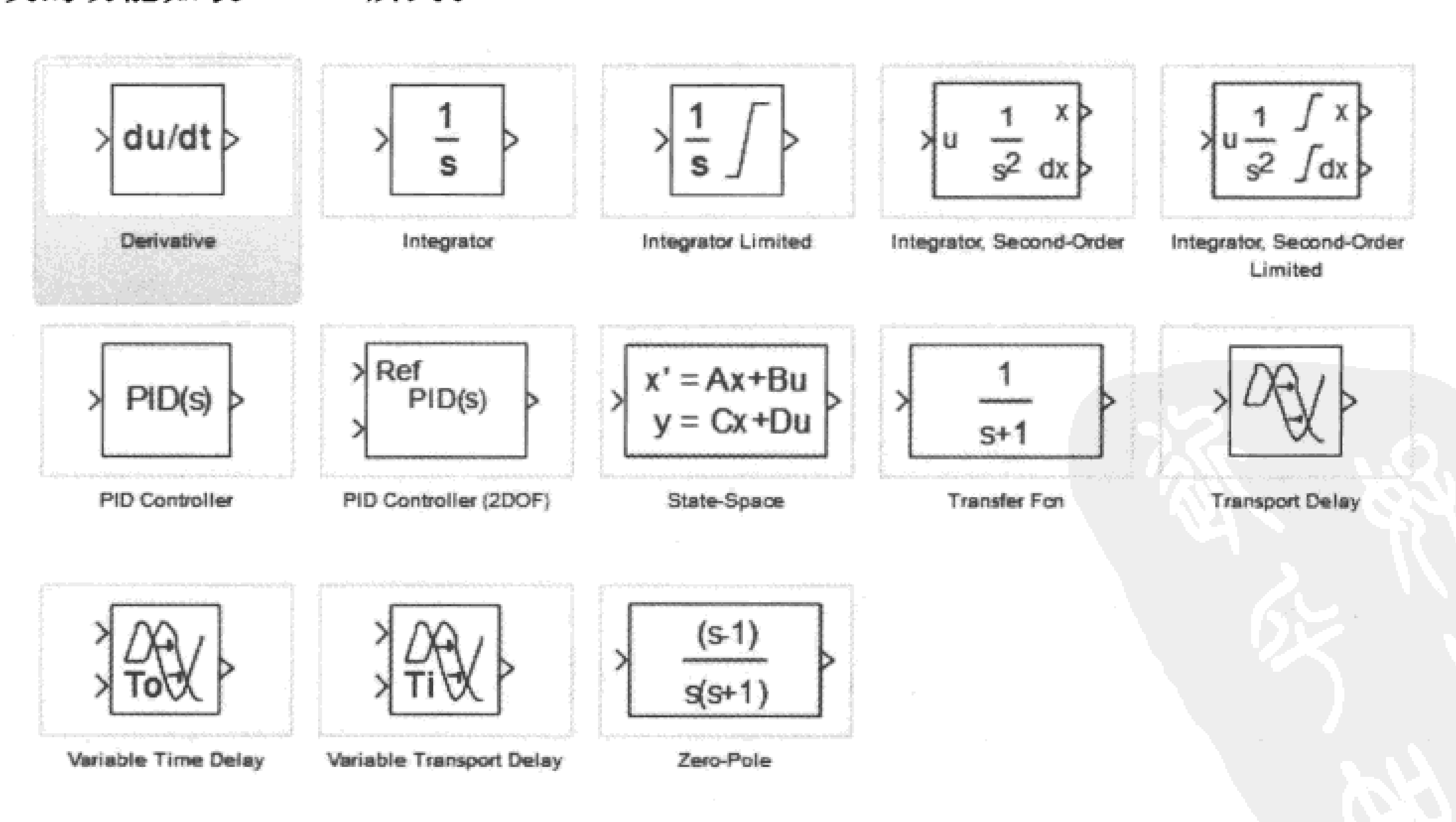


图 2.1.4 连续模块库

表 2.1.3 连续模块库列表

名 称	功 能	名 称	功 能
Derivative	微分	Integrator	积分
Integrator Limited	有限积分	Integrator 2nd-order	二阶积分
Integrator 2nd-order Limited	二阶有限积分	PID Controller	比例微积分控制器
PID Controller(2DOF)	双自由度比例微积分控制器	State-Space	状态空间
Transfer Fcn	传递函数	Transport Delay	时间延迟
Variable Time Delay	可变时间延迟	Variable Transport Delay	可变时间延迟
Zero - Pole	零极点		

结合本书是讲述基于模型设计的思想开发 MCU 器件，本章将以 Simulink 在控制电动机中的应用为例，介绍 Simulink 的建模与调试技术。这里值得一提的是 PID 控制模块。PID 控制器就是根据系统的误差，利用比例、积分、微分计算出控制量进行控制的。它是在较新版本的 Simulink 中才新增并逐步完善的一个模块，R2010b 版已经具备自动调节功能。具体原理和使用将在后面分析。

3. 离散模块库(discontinuous)

离散模块在涉及数字信号系统中被广泛使用，基于这种考虑，mathworks 公司单独列出了离散系统模块库。离散模块库中的模块和其功能如图 2.1.5 所示。

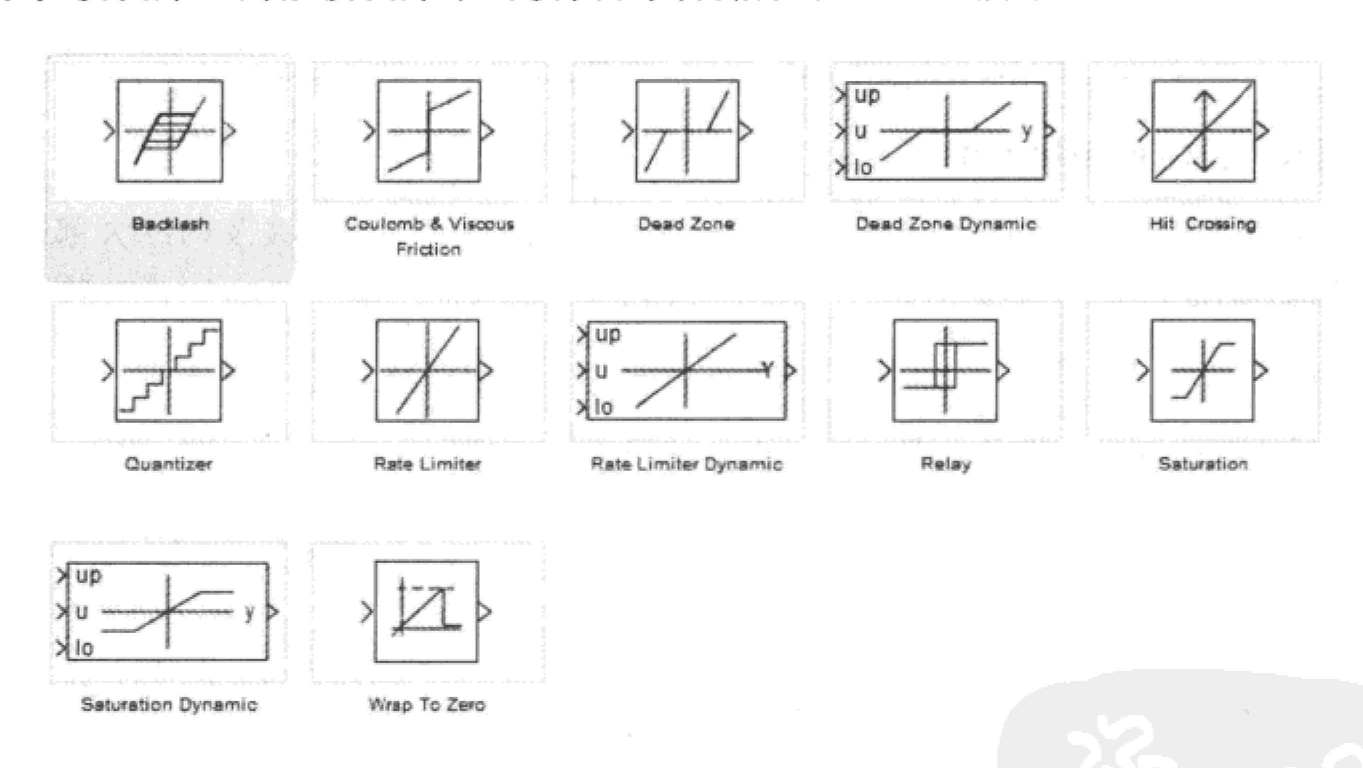
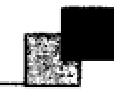


图 2.1.5 离散模块库

其中常用模块的功能如表 2.1.4 所列。

表 2.1.4 离散模块库列表

名 称	功 能	名 称	功 能
Difference	差分	Discrete Derivative	离散微分方程
Discrete FIR Filter	离散 FIR 滤波器	Discrete Filter	离散滤波器



续表 2.1.4

名 称	功 能	名 称	功 能
Discrete PID Controller	离散 PID 控制器	Discrete PID Controller (2DOF)	离散双自由度 PID 控制器
Discrete State-Space	离散状态空间	Discrete Transfer Fcn	离散传递函数
Discrete Zero-Pole	离散零极点	Discrete Time Integrator	离散时间积分
1st-order Hold	一阶保持器	Integer Delay	采样保持
Memory	记忆	Tapped Delay	采样周期延迟
Transfer Fcn 1st-order	一阶传递函数	Transfer Fcn Lead or Lag	传递函数(超前或延迟)
Transfer Fcn Real Zero	传递函数(有零点无极点)	Unit Delay	单位延迟
Zero-Order Hold	零阶保持器		

4. 数学运算模块库(math operations)

数学运算模块将很多数学运算封装成模块的形式,使数学运算操作大大简化,减少了很多程序设计上的烦琐过程。此模块库所包含的模块如图 2.1.6 所示。

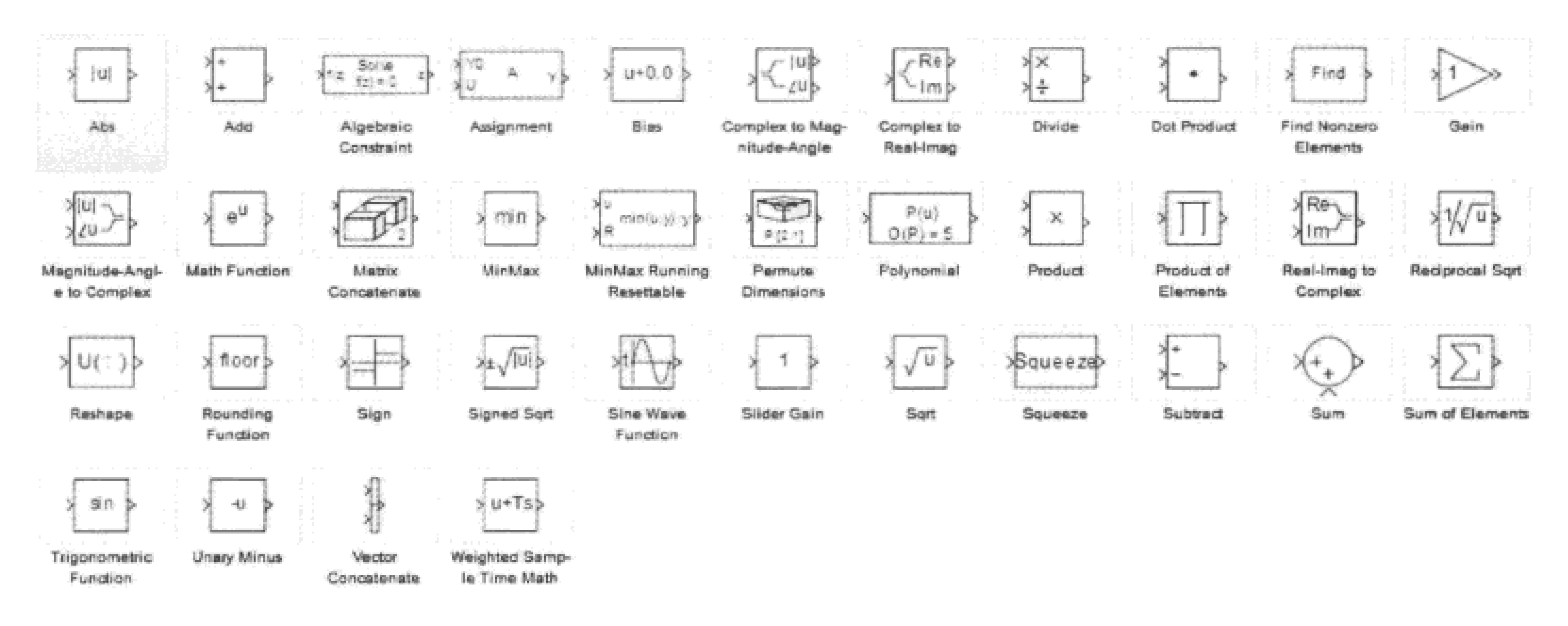


图 2.1.6 数学运算模块库

其中常用模块的功能如表 2.1.5 所列。

表 2.1.5 数学运算模块库列表

名 称	功 能	名 称	功 能
Sum	对输入求代数和	Rounding Function	取整
Gain	常量增益	MinMax	求最值
Slider Gain	可用滑动条改变的增益	Trigonometric Function	三角函数
Product	对输入求积或商	Algebraic Constraint	强制输入信号为 0
Dot Product	点积	Complex to Magnitude - Angle	复数的幅值相角
Sign	取输入的正负符号	Magnitude - Angle to Complex	根据幅值相角得到复数
Abs	绝对值(模)	Complex to Real - Imag	复数的实部虚部
Math Function	数学运算函数	Real - Imag to complex	由实部虚部求复数

5. 信号源模块库(signal attributes)

信号源模块库如图 2.1.7 所示。

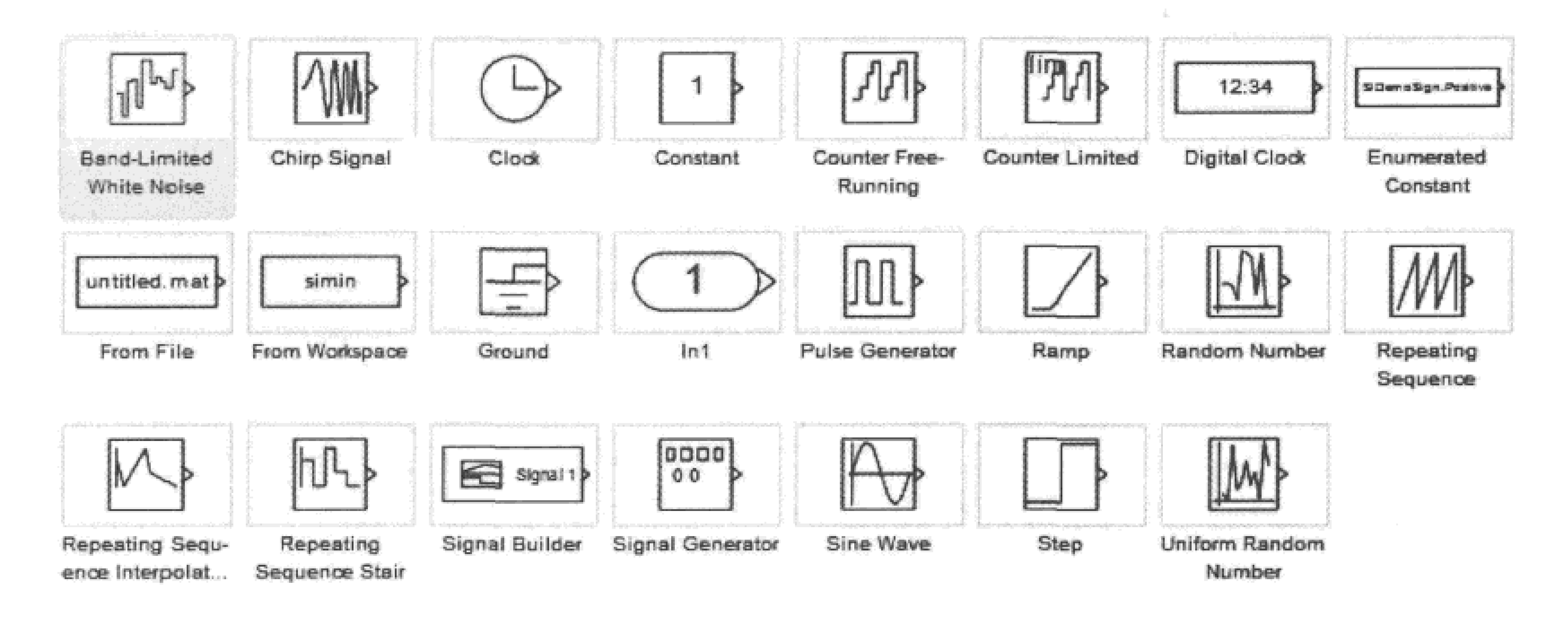


图 2.1.7 信号源模块库

其中常用模块的功能如表 2.1.6 所列。

表 2.1.6 信号源模块库列表

名 称	功 能	名 称	功 能
Band－Limited White Noise	限带白噪声	Chirp Signal	频率变化的正弦信号
Clock	时钟信号	Constant	常数
Counter Limited	受限计数器	Digital Clock	数字时钟
Enumerated Constant	枚举常数	From File	从文件读数据
From WorkSpace	从工作空间读数据	Ground	接地
Inport	输入接口	Pulse Generator	脉冲发生器
Ramp	线性增或减的信号	Random Number	随机数
Repeating Sequence	重复系列	Repeating Sequence Interpolated	重复序列插值
Repeating Sequence Stair	阶梯状重复序列	Signal Builder	产生分段线性的可交替信号
Signal Generator	信号发生器	Sine Wave	正弦信号
Step	阶跃信号	Uniform Random Number	平均分布的随机信号

6. 信号接收模块库(sinks)

信号接收模块库如图 2.1.8 所示。

其中常用模块的功能如表 2.1.7 所列。

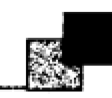




图 2.1.8　信号接收模块库

表 2.1.7　信号接收模块库列表

名　称	功　能	名　称	功　能
Display	显示输入值	Output	输出端口
Scope and Floating Scope	显示仿真信号	Stop Simulation	非零时停止仿真
Terminator	终止输出信号	To File	输出到文件
To WorkSpace	输出到工作空间	XY Gragh	作图

7. 用户自定义模块库(user-defined functions)

用户自定义模块库如图 2.1.9 所示。

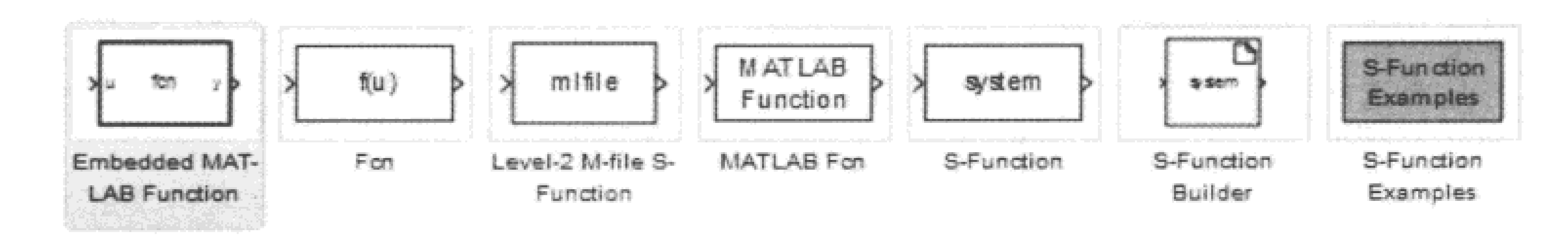


图 2.1.9　用户自定义模块库

其中常用模块的功能如表 2.1.8 所列。

表 2.1.8　用户自定义模块库列表

名　称	功　能	名　称	功　能
Embedded MATLAB Function	嵌入式 MATLAB 函数	Fcn	各种函数组合
Level-2 M-File S-Function	二级 M 文件 S 函数	MATLAB Fcn	使用 MATLAB 函数作为输入
S -Function	S 函数	S-Function Builder	S 函数构造器

2.1.3　模块的基本操作

1. 模块的查找

在模块库页面的左侧以树形结构显示了一系列的子库，如连续模块库、离散模块库、数学运算模块库，当用户对这些模块比较熟悉以后，可以非常方便、快捷地找到自己需要的模块。

如果是刚刚接触 Simulink 的用户，这里还提供了查找功能，方便不熟悉它的用户使用。

如图 2.1.10 所示。

当用户找到自己所需要的模块后，就要把它复制到模型编辑窗口中以便进行下一步的操作。这里可以直接把模块拖拽到编辑窗口中完成复制，也可以右击，在弹出右键菜单中选择 Add to new model 命令，如图 2.1.11 所示。

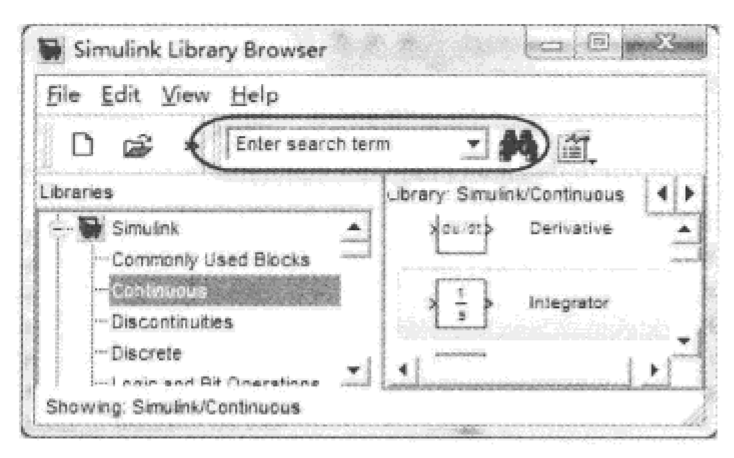


图 2.1.10 模块查找

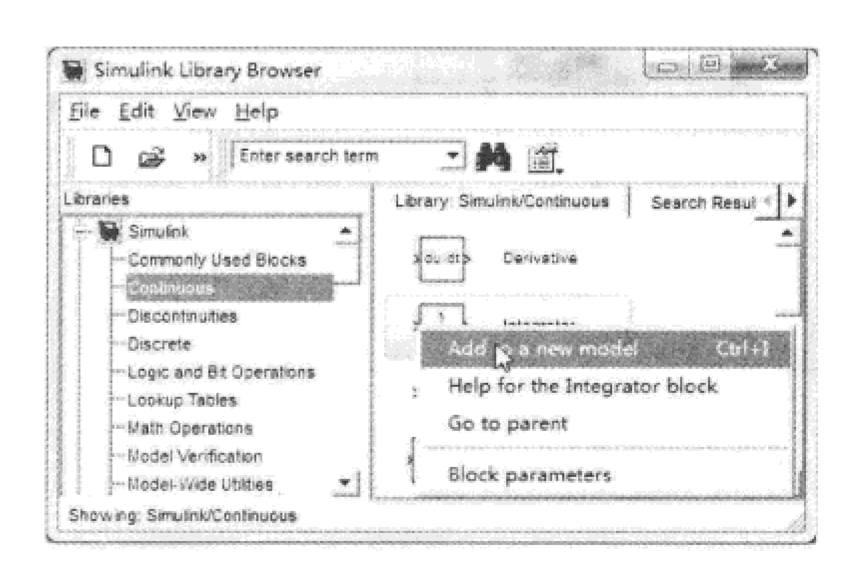


图 2.1.11 添加模块

2. 模块的选定

模块被加入到编辑窗口以后，要通过选定模块才能对其进行操作。选定单个模块可以直接单击目标模块；也可以按住鼠标任意键拖动，此时会出现一个虚线框，当虚线框包围了目标模块时松开鼠标，即可选中模块。

若是要选中多个模块就需要用上述的虚线框法来操作，如图 2.1.12 所示。选择编辑窗口中所有对象的方法是单击菜单 Edit→Select All 命令。

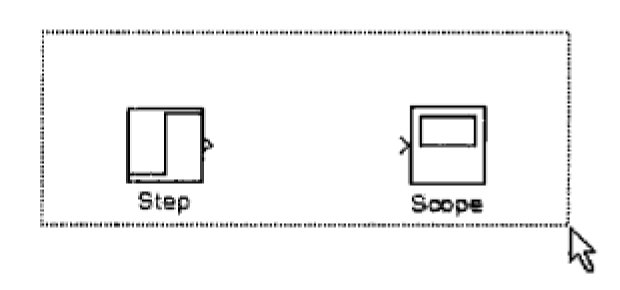


图 2.1.12 选定一组对象

3. 模块的连接和调整

(1) 模块的调整。为了使模型更加美观和符合逻辑，有时需要对模块的大小、方向等作适当的修改。模块大小的调整。选中模块，用鼠标选中模块周围 4 个黑色方块中的任意一个开始拖动，这时会出现一个虚线框表示的新模块大小示意，调整至需要的大小后松开鼠标即可，如图 2.1.13 所示。

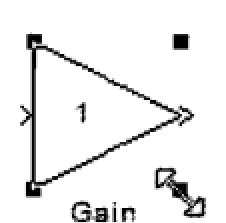
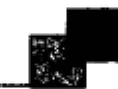


图 2.1.13 调整模块大小

模块的旋转。选中模块想要旋转的模块，在菜单中选择 Format 命令，在次级目录中选择 Flip Block 命令可以使模块 180°旋转；选择 Rotate Block 使模块旋转 90°。如图 2.1.14 所示。

(2) 模块的连接。模型中的信号是从模块经连线传输到下一个模块的，因此模块间的连线被称为信号线，当模块设置好以后，只有将它们按一定的顺序连接起来才能完成一个正确的模型。



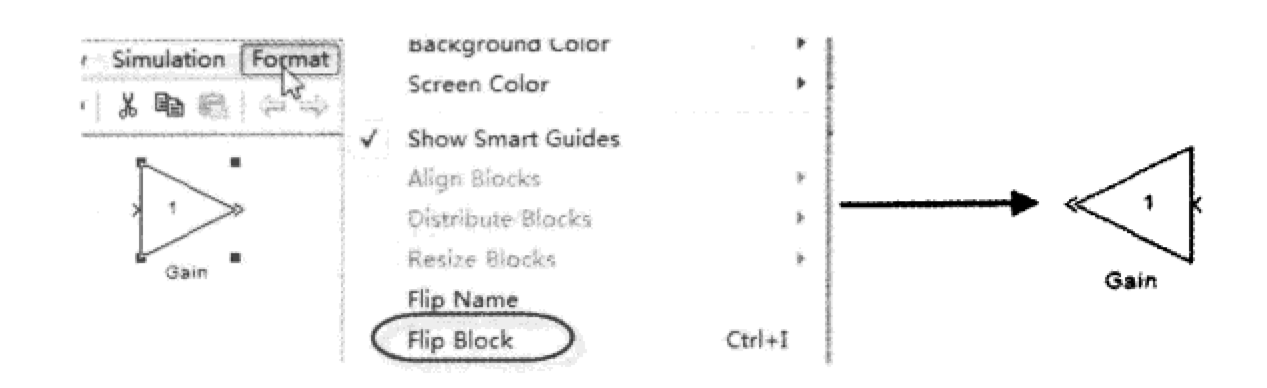


图 2.1.14　模块翻转

① 自动连线：Simulink 系统具备自动连线的功能，步骤如下：选择一个具有信号输出的模块，按 Ctrl 键，然后单击要连接到的模块，如图 2.1.15 所示。

② 手动连线：用户也可以选择自己手动连线。把鼠标指针移动到第一个模块的输出端口，这时鼠标指针的形状会变为一个十字，如图 2.1.16 所示。



图 2.1.15　自动连线　　　　图 2.1.16　手动连线

按下并拖动鼠标至目标模块的输入端口处，这时鼠标指针会变成一个双十字，松开鼠标后两个模块就连接起来了，如图 2.1.17 所示。

提示：用手动方法连接模块时是可以从输出到输入口画连接线的。

③ 分支连线：是指从一条已经存在的连线上另外再拉出一条线用来连接一个模块的输入口。这时，原连线和分支连线上传输的是同一个信号。

把鼠标指针移动到原有连线上，按 Ctrl 键，拖动鼠标，使鼠标指针到目标模块的端口，松开鼠标，完成连线，如图 2.1.18 所示。

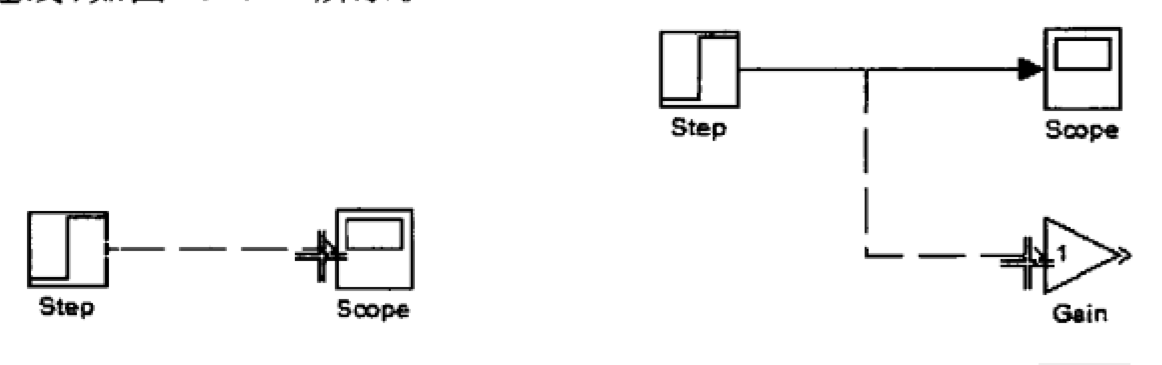


图 2.1.17　手动连线　　　　图 2.1.18　分支连线

④ 移动连线：把鼠标指针移动到想要移动的连线处，按住鼠标左键并拖动，使鼠标指针到目标位置，松开鼠标，完成移动，如图 2.1.19 所示。

⑤ 连线的折曲：选择要折曲的线段，将鼠标指针移动到要折曲的点上，按 shift 键，并按住鼠标左键，这时鼠标指针会变成一个圆圈状，移动折曲点至目标位置后松开鼠标，完成折曲。如图 2.1.20 所示。

⑥ 连线中插入模块：如果模块只有一个输入口和输出口，那么可以将该模块直接插入到一天连线中，如图 2.1.21 所示。

⑦ 连线的注释：有时为了增加模型的可读性，常常会给信号线加上注释。在要添加注释

的连线附近双击会出现一个文本输入框，用户可以根据需要添加适当的注释，注释的位置是可以通过鼠标拖拽更改位置的，如图 2.1.22 所示。

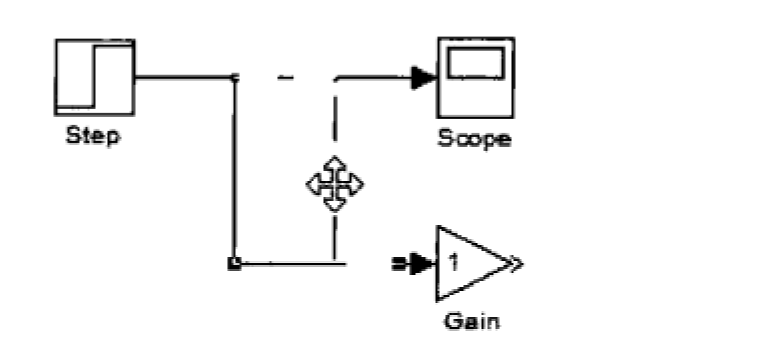


图 2.1.19　移动连线

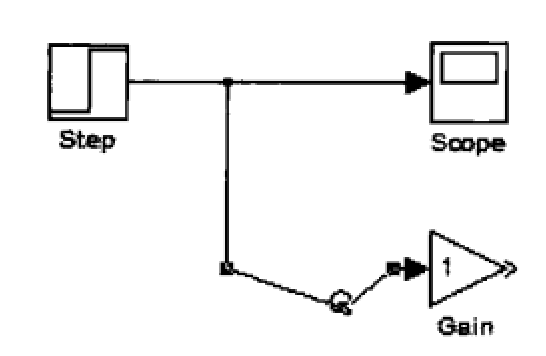


图 2.1.20　曲折连线

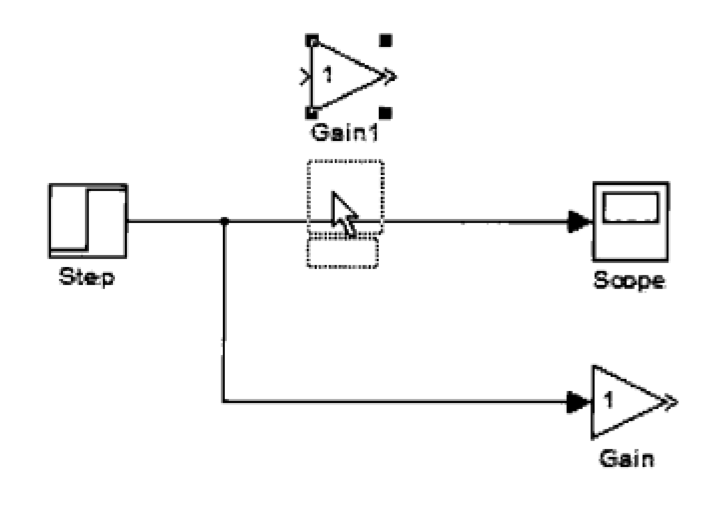


图 2.1.21　连线中插入模块

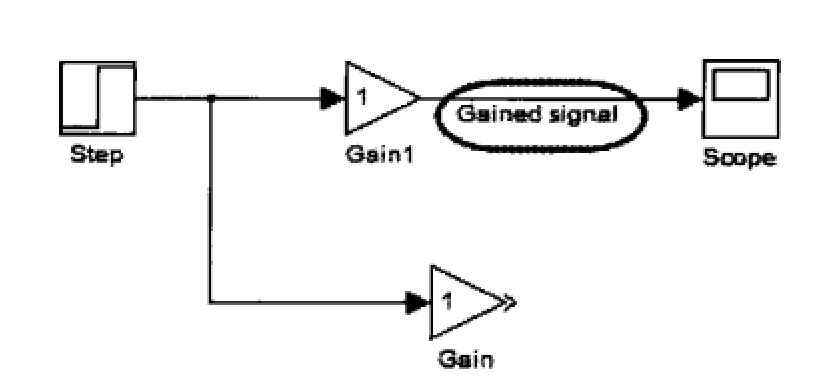


图 2.1.22　连线的注释

2.2　搭建直流电动机模型

直流电动机是控制系统中的常用组件，它给系统提供旋转动力或者和其他组件配合进行传动。下面将通过一个直流电动机模型(DC Motor)来介绍如何在 Simulink 中建立 LTI(Linear time-invariant)系统并实现对其转速的控制。

2.2.1　数学模型分析

电动机可分为电气部件和机械部件两部分，电动机模型如图 2.2.1 所示。

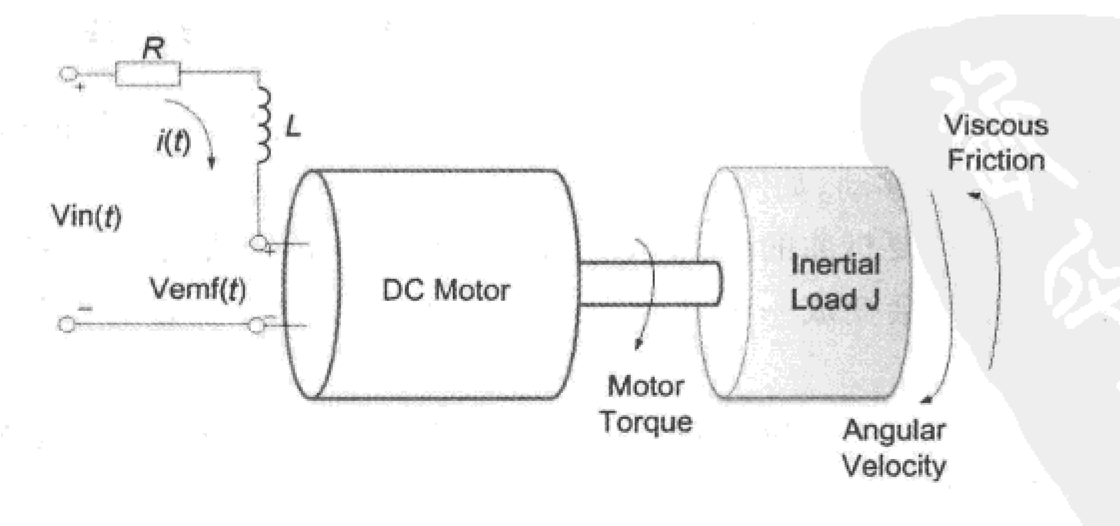
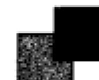


图 2.2.1　电动机模型



电动机的电气部分和机械部分是通过转子联系起来的，转子即为电能与机械能转化的枢纽。这两部分的方程都要受到转子感应电动势的影响。

对于左边的电气部分，由电磁感应定律可知，转子会产生一个反向的感应电动势来阻止转子的运转，外界电源 V_{in} 要克服反向感应电动势做功，才能使转子转动。我们可以先把其他元件抽象为电阻和电感，把精力集中到主要矛盾上来：电动机如何平稳运转。

当电动机正常运转时，先分析电气部分回路：外部电源 V_{in}、电阻和电感上的电压降 V_R 和 V_L、转子上的反向感应电动势 V_{emf} 在同一个回路中。由电路分析的基础知识 KVL 定律（基尔霍夫电压定律）可以得到：

$$V_{in}=V_R+V_L+V_{emf} \tag{2-1}$$

对于右边的机械部分，稍复杂一些。当电动机带着负载匀速旋转时，其输出转矩必定与负载转矩相等。在非理想电动机模型中（即不忽略机械、电磁损耗），轴承的摩擦、电刷和换向器的摩擦、转子铁心中的涡流、磁滞损耗都要引起阻转矩。此阻转矩用 T_f 表示。这样，电动机的输出转矩便等于电磁转矩 T 减去电动机本身的阻转矩 T_f。所以，当电动机克服负载阻转矩 T_L 匀速旋转时，则有：

$$T_L=T_O+T-T_f \tag{2-2}$$

实际上，电动机经常运行在转速变化的情况下，例如启动、停转或反转等，因此必须讨论转速改变时的转矩平衡关系。当电动机的转速改变时，由于电动机及负载具有转动惯量，将产生惯性转矩 T_j，则有 $T_j=J\times\frac{d\omega}{dt}$。其中 J 是负载和电动机的转动惯量，ω 是电动机的角速度，$\frac{d\omega}{dt}$ 是其角加速度。

为了使模型能覆盖这种更一般的情况，可根据牛顿力学定律对式(2-2)做一些微调：

$$T_O-T_L=T_j=J\times\frac{d\omega}{dt} \tag{2-3}$$

得到表示电动机基本原理的方程后，需要设置一些参数以备建模使用。如：电阻 R，电感 L，转动惯量 J，转矩 T_m，旋转角度 θ，转矩常量 K_m，感应电动势常量 K_{emf}，粘滞摩擦因数 K_f。

因为 $\omega(t)=\dot{\theta}$，$V_{emf}=K_{emf}\omega(t)$，式(2-1)可化为：

$$L\frac{dj}{dt}+Ri=V_{in}-V_{emf} \tag{2-4}$$

又因为 $\ddot{\theta}=\frac{d\omega}{dt}$，式(2-3)可化为：

$$J\times\ddot{\theta}+K_f\dot{\theta}=T_m \tag{2-5}$$

将式(2-4)、式(2-5)两式子用电压 V_{in} 和转子角速度 ω 表示出来可写为：

$$\frac{dj}{dt}=\frac{V_{in}(t)-Ri(t)-K_{emf}\omega(t)}{L} \tag{2-6}$$

$$\frac{d\omega}{dt}=\frac{K_m i(t)-K_f\omega(t)}{J} \tag{2-7}$$

为了表示方便，对上式做拉普拉斯变换，转换到 S 域。

根据拉普拉斯变换的规则，在 0 初始条件下，一阶导数→s；二阶导数→s^2，则可得到：

$$I(s)=\frac{V_{in}(s)-RI(s)-K_{emf}\omega(s)}{L} \tag{2-8}$$



$$\omega(s)=\frac{K_m I(s)-K_f\omega(s)}{J} \tag{2-9}$$

合并化简后得到系统传输函数：

$$\frac{\omega(s)}{V_{in}(s)}=\frac{K_m}{JLs^2+(K_fL+JR)s+(K_fR+K_{emf}K_m)} \tag{2-10}$$

至此，分析了直流电动机的传递函数，下面将利用 Simulink 建立直流电动机模型。

2.2.2 模型搭建与参数设置

1. 模型搭建

直流电动机的模型需要用到 source 库中的 step 模块，sinks 库中的 scope 模块，math operations库中的 add、gain，和 continuous 库中的 integrator。

（1）添加 source 库中的 step 模块到模型中，如图 2.2.2 所示。

默认情况下，信号从第一秒处产生跃变。

（2）添加 sinks 库中的 scope 模块到模型中，如图 2.2.3 所示。

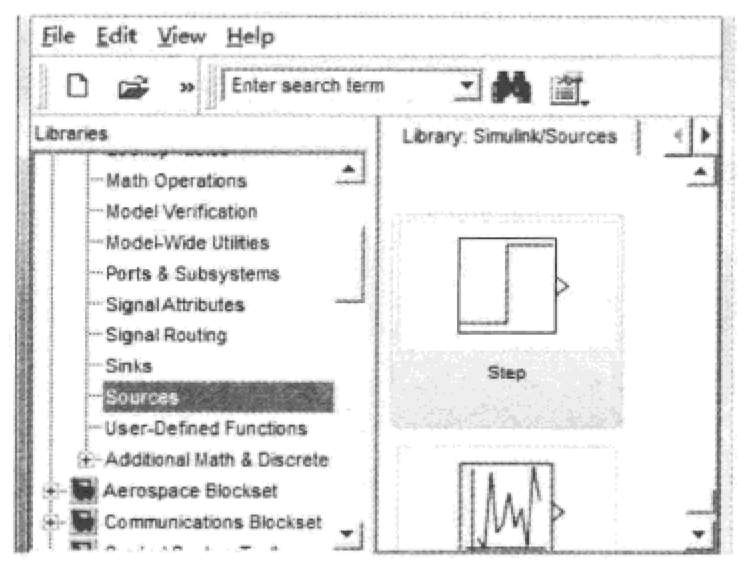



图 2.2.2 step 模块

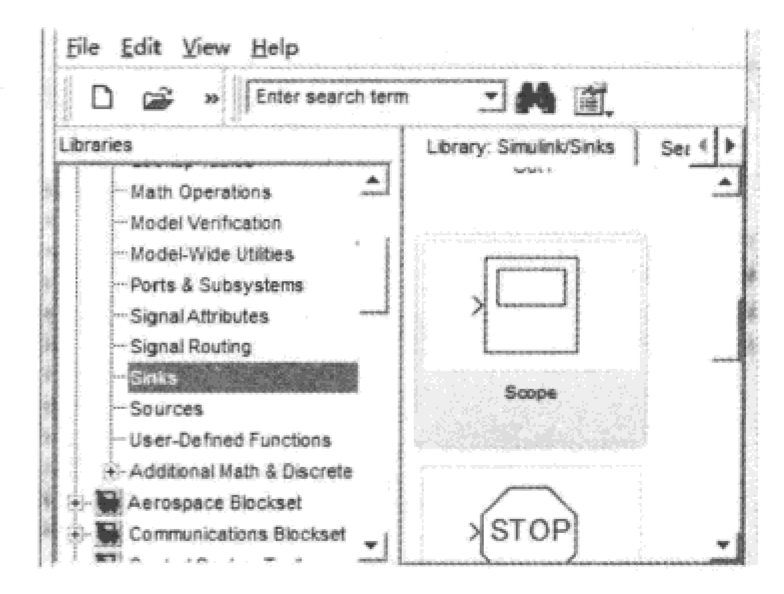


图 2.2.3 scope 模块

示波器模块是可以同时显示多个输入信号的。例如，双击打开示波器模块，单击按钮 ，在 number of axes 中输入参数 3 即可使示波器变为 3 个输入口，如图 2.2.4 所示。

（3）添加 math operations 库中的 add 模块到模型中，如图 2.2.5 所示。

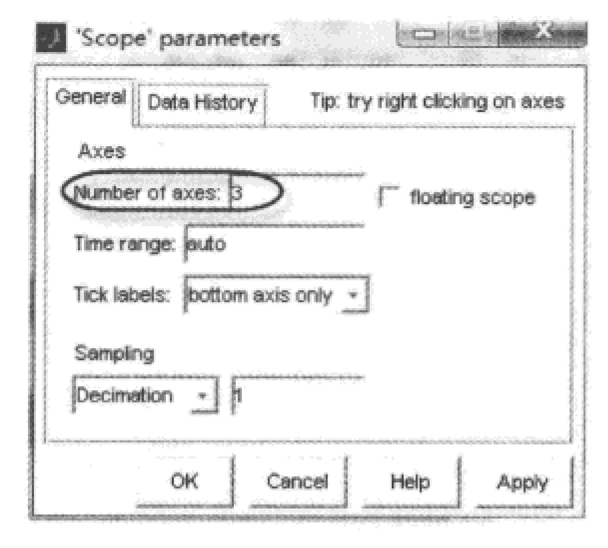


图 2.2.4 设置 scope 模块

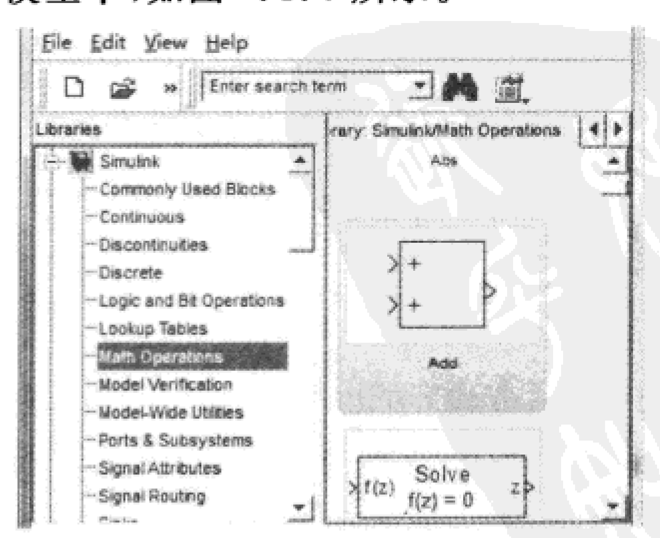
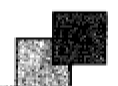


图 2.2.5 add 模块



为了正确表达电动机模型的系统方程，需要两个 Add 模块，其符号参数 List of signs 需要分别设置为"＋－－"和"＋－"，这样 Add 模块会分别含有 2 个和 3 个输入接口。如图 2.2.6 所示。

(4) 添加 math operations 库中的 gain 模块到模型中，如图 2.2.7 所示。

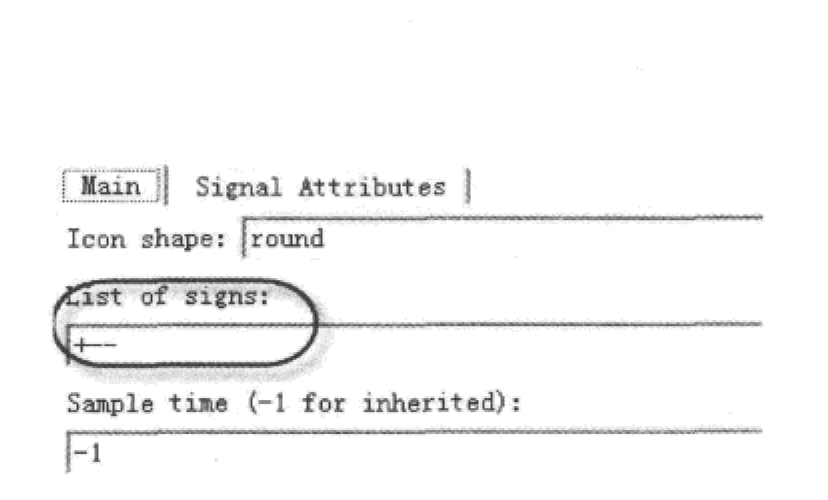


图 2.2.6　设置 add 模块

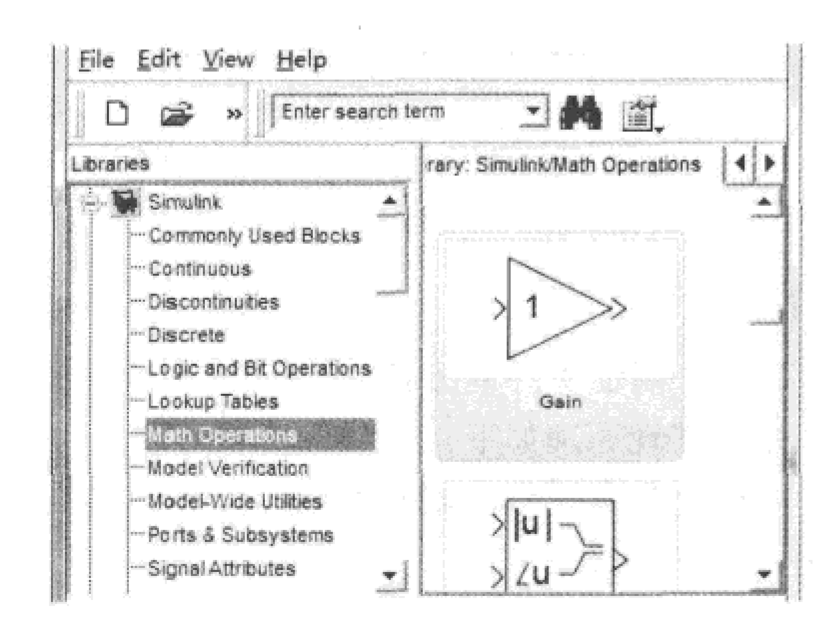


图 2.2.7　gain 模块

gain 模块的参数需要修改。该模型需要 6 个 gain 模块，参数分别设置为：1/L、Km、1/J、R、Kf、Kemf。可以通过双击放大器模块打开其参数设置页面，然后输入参数，如图 2.2.8 所示。

(5) 添加 continuous 库中的 integrator 模块到模型中，如图 2.2.9 所示。

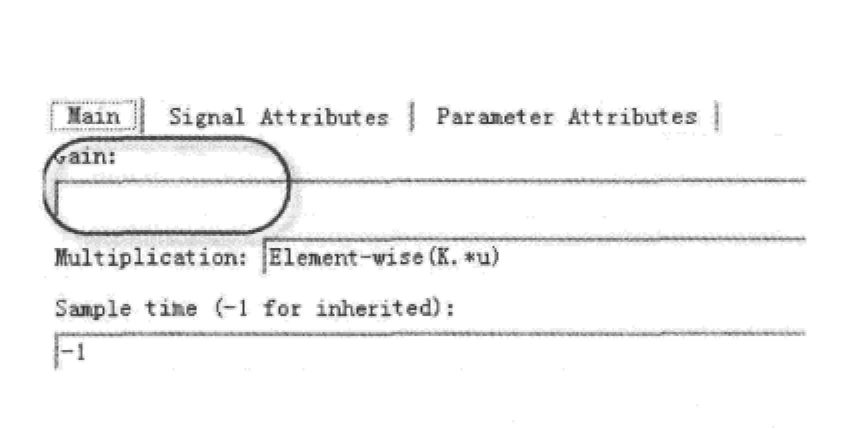


图 2.2.8　设置 gain 模块

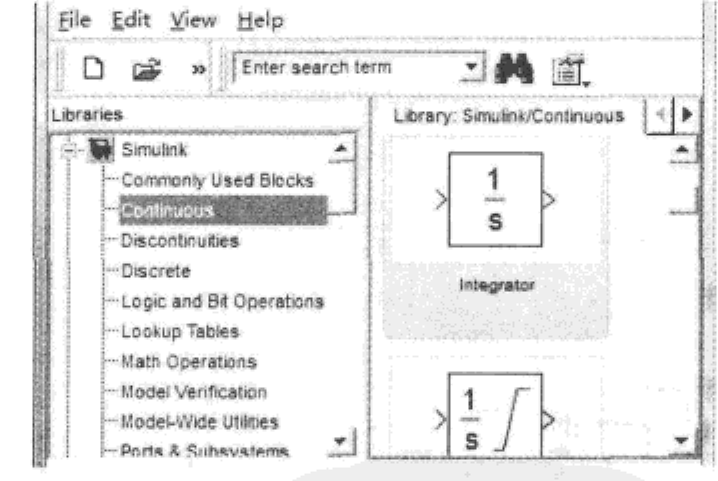


图 2.2.9　integrator 模块

模型中需要两个 Integrator 模块，参数可采用默认值。

(6) 按照式(2－10)表达的逻辑，在第一个积分器之前算子为 S^2，其系数为 JL；第一个和第二个积分器之间算子为 S，其系数为(K_fL+JR)；第二个积分器之后算子为 1，其系数为($K_fR+K_{emf}K_m$)。由此可以连接其中的各个模块，得到图 2.2.10 所示的模型。

左侧的模块对应于式(2－8)，其输出为转矩 Torque；右侧的模块对应于式(2－9)，其输出为转速 $\omega(t)$。此模型实现了系统传输函数所描述的由输入电压控制直流电动机转速的功能。当输入信号为 Step 阶跃信号时，输出为其阶跃响应；当输入为外部实际控制信号时，电动机转

速 $\omega(t)$ 即由输入信号所控制。

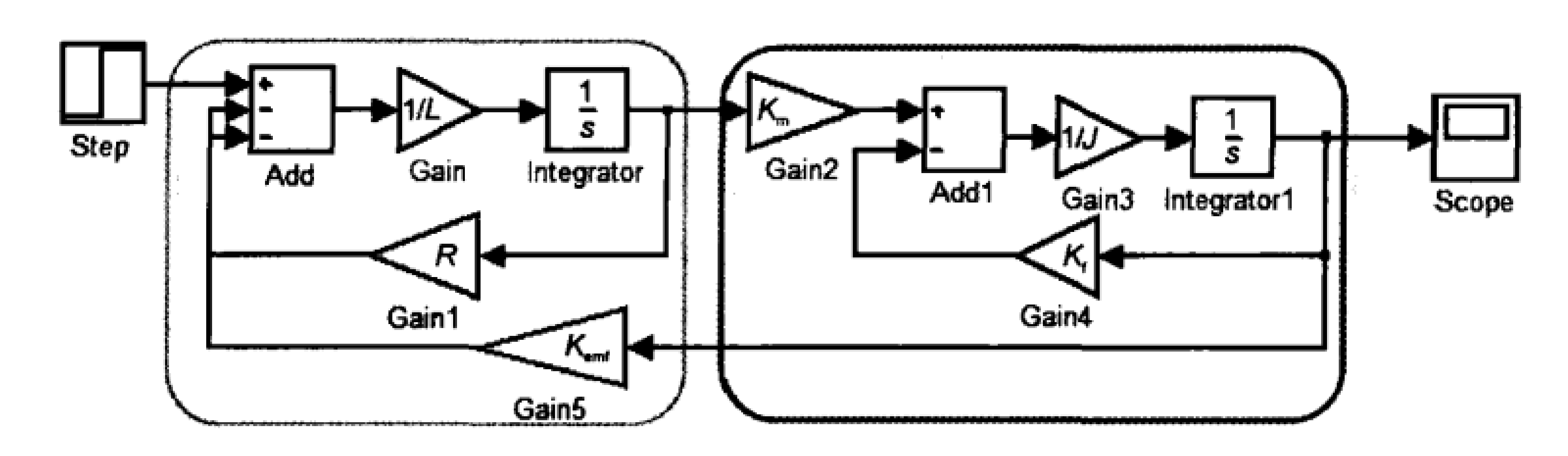


图 2.2.10 Simulink 电动机模型

为了后面更好地分析电动机模型，在仿真时同时输出其转动控制信号和输出转速、角度和转矩。实现原理如下：

(1) 控制信号与输出转速比较在添加 Signal Routing 库中的 Mux 模块到模型中，如图 2.2.11所示。

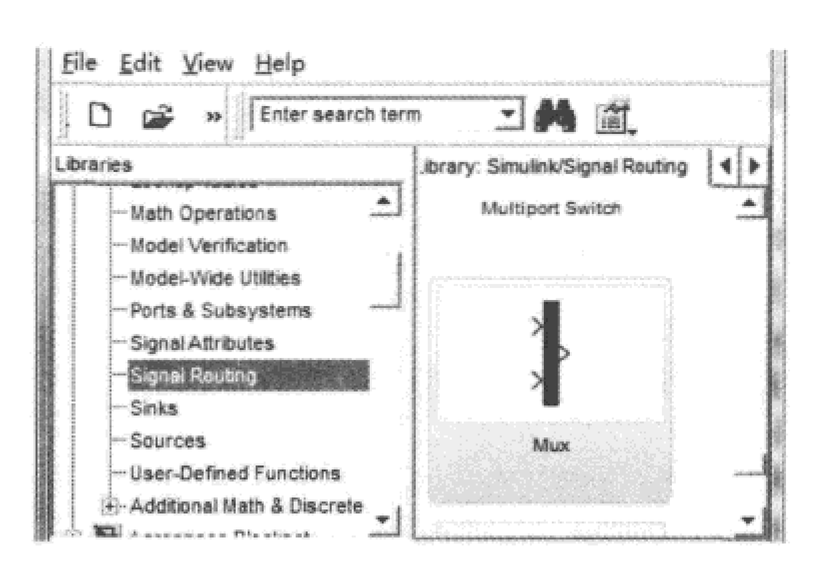


图 2.2.11 Mux 模块

修改模型如图 2.2.12 所示。

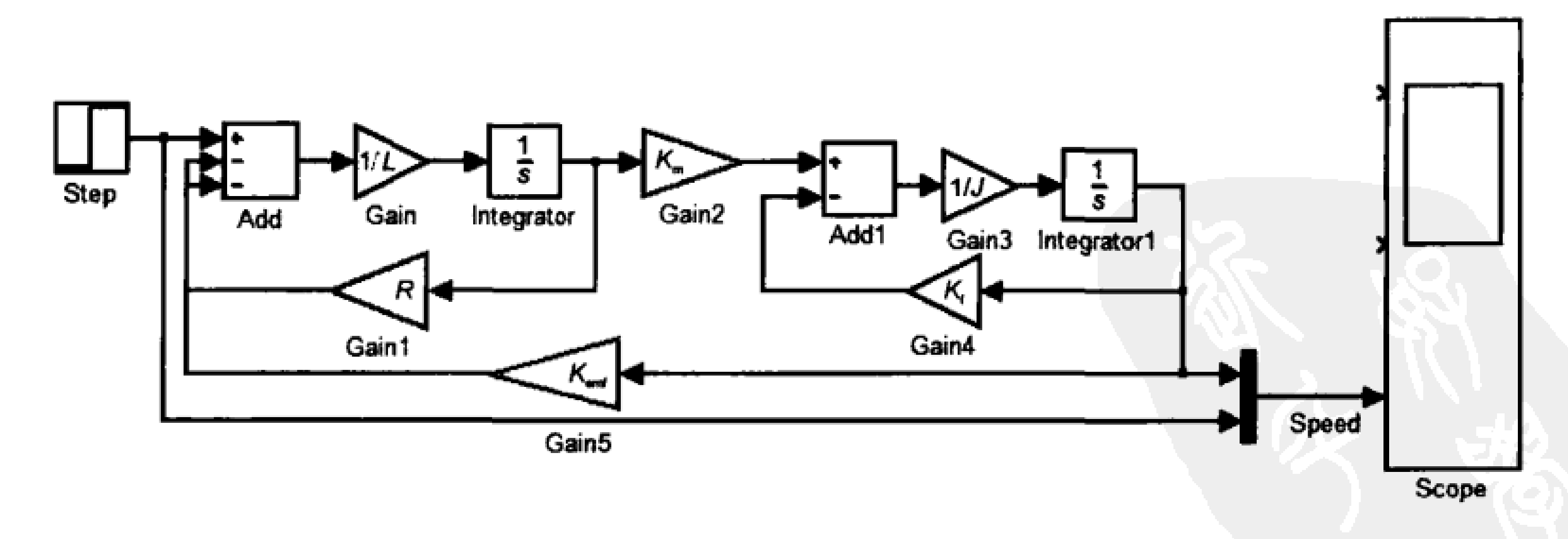


图 2.2.12 Simulink 电动机模型

(2) 转矩信号输出如图 2.2.10 所示，左边框图模块组的输出既是转矩信号。

(3) 角度信号输出由电动机数学模型公式：



$$\frac{\omega(s)}{V_{in}(s)}=\frac{K_m}{JLs^2+(K_fL+JR)s+(K_fR+K_{emf}K_m)} \tag{2-11}$$

可知该模型的输出为 $\omega(t)$，即旋转角速度，且 $\theta=\int_0^t w(\tau)\mathrm{d}\tau$，因此，模型的输出经过一个积分器即转化为角度，修改模型如图 2.2.13 所示。

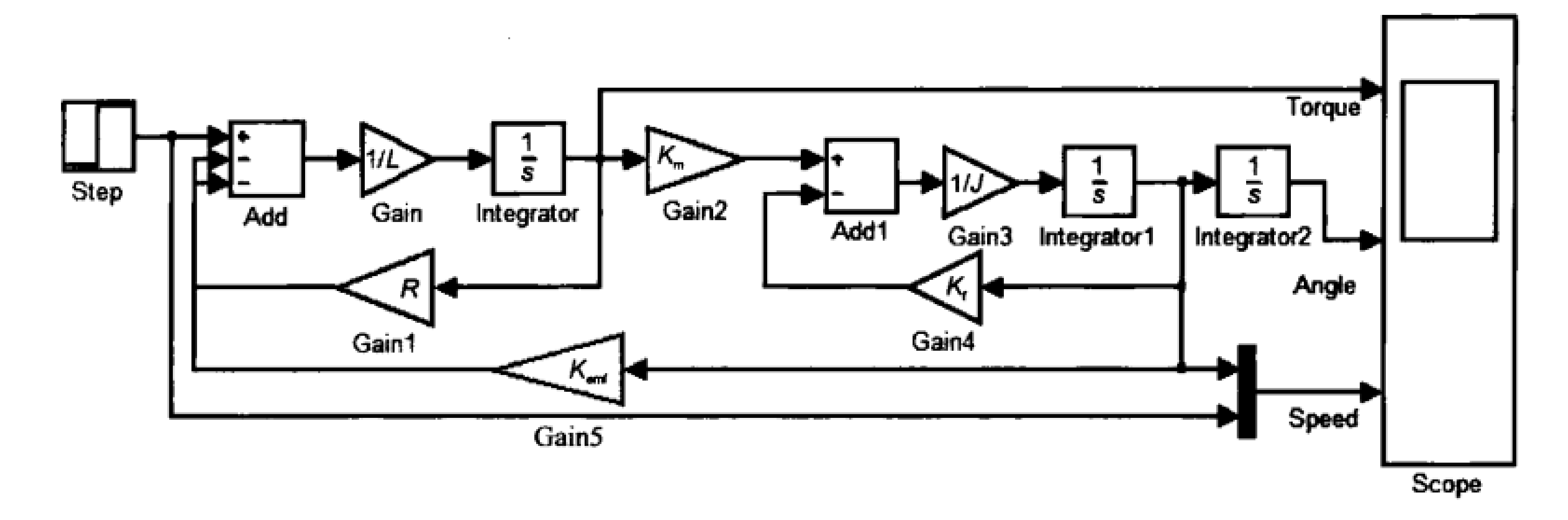


图 2.2.13　Simulink 电动机模型

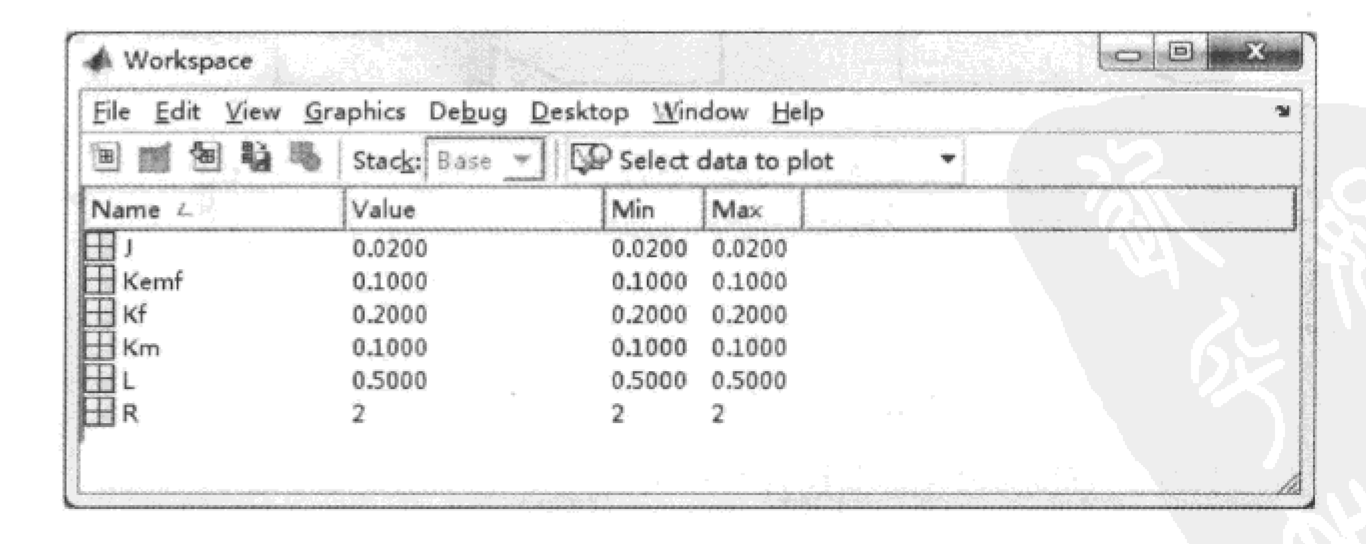
2. 参数设置

最后，在仿真之前一定要设置模型仿真参数，如仿真时间，步长，表达式中定义的 R、L、J 等参数值等。

(1) 在 MATLAB 中以 M 文件或命令行定义变量。用户可以直接在 MATLAB 命令窗口中输入以下命令：

```
>> R = 2;              % 电阻值 Resistor
>> L = 0.5;            % 电感值 Inductor
>> Km = 0.1;           % 转矩常量 Torque Constant
>> Kemf = 0.1;         % 反电动势常量 Back EMF Voltage Constant
>> Kf = 0.2;           % 阻滞系数常量 Viscous Friction Constant
>> J = 0.02;           % 惯性载荷 Inertial Load
```

完成后，在 MATLAB 工作空间中可以看到这些变量，如图 2.2.14 所示。



Name	Value	Min	Max
J	0.0200	0.0200	0.0200
Kemf	0.1000	0.1000	0.1000
Kf	0.2000	0.2000	0.2000
Km	0.1000	0.1000	0.1000
L	0.5000	0.5000	0.5000
R	2	2	2

图 2.2.14　仿真参数设置



这样就完成了模型中以表达式形式定义的参数的赋值。为了方便以后再次运行模型，可以把这些数据保存下来，在运行模型之前装载这些变量即可。

全选这些变量，单击按钮，输入保存名 DC_motor(用户可自定义)，扩展名为. mat，即把数据保存到了当前目录下，以后可以通过双击直接装载变量，如图 2.2.15 所示。

当然，用户也可以将上述命令窗口中的指令保存为 M 文件的形式，并在运行模型前先执行 M 文件来装载变量，如图 2.2.16 所示。

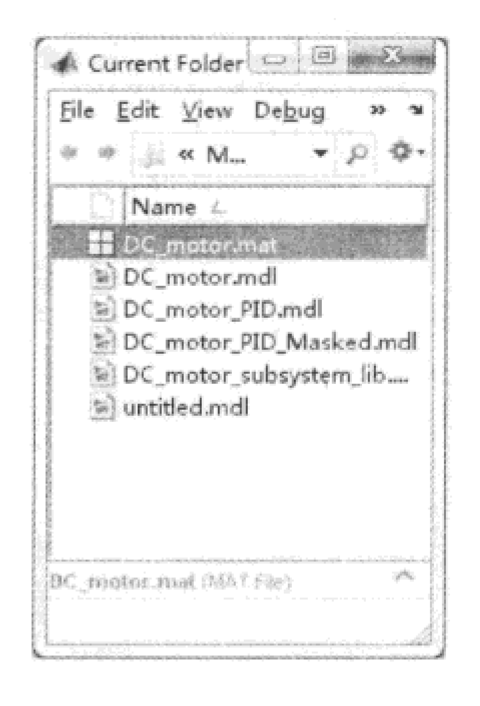


图 2.2.15 保存参数

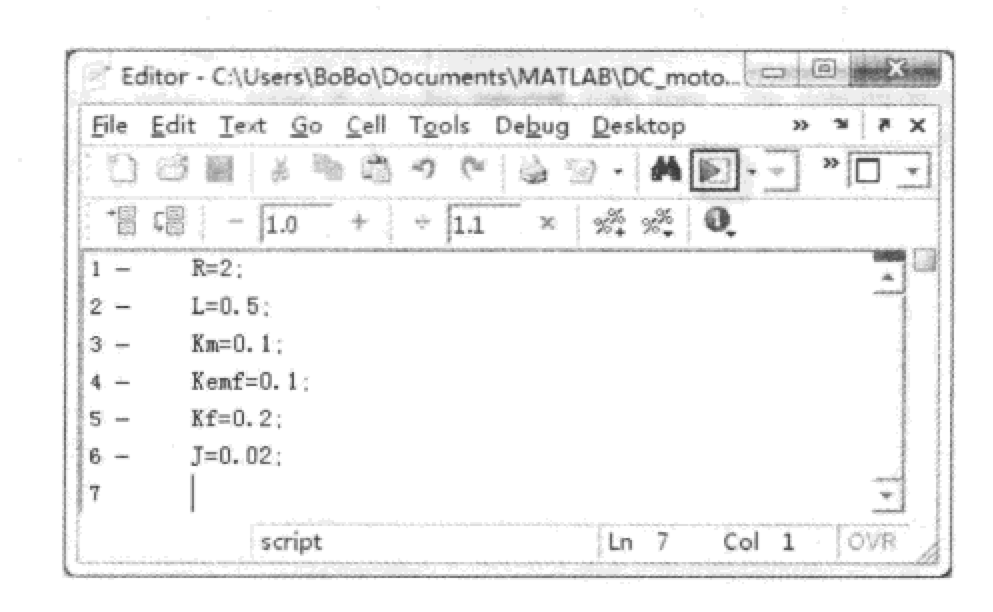


图 2.2.16

(2) 在 Simulink 中用 Annotation 方式定义变量。通过在 Simulink 中定义 Annotation，可以直接在模型编辑窗口中完成对参数的赋值，而不必再从 MATLAB 中读取. mat 文件和 M 文件，省去了不少麻烦。

在模型的任意空白处双击会出现一个可编辑的文本框，写入相关文字，如 Parameters Configuration。如图 2.2.17 所示。

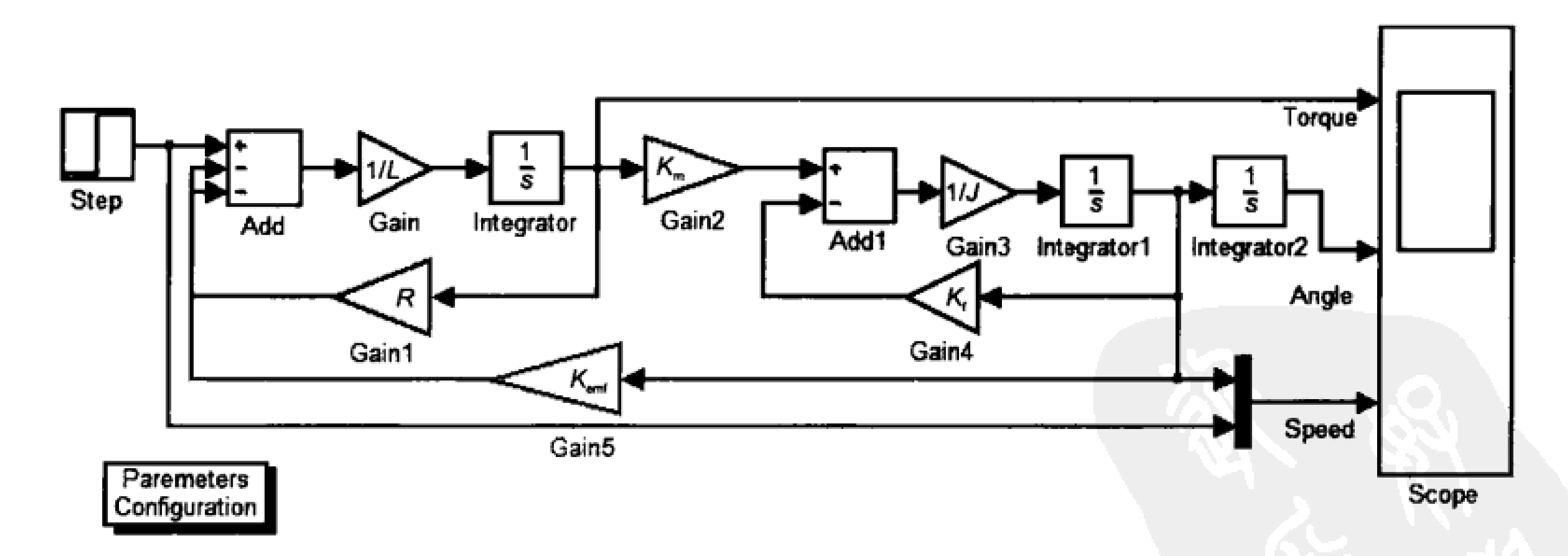
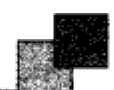


图 2.2.17 添加 Annotation

右击文本框会弹出右键菜单，其中较为有用的有如下几项命令：

① Front...：调节字体字号。

② Text Aligment：文本对齐方式。



③ Hide/Show Drop Shadow：隐藏/显示阴影。

④ Annotation Properties... Annotation：属性设置。

⑤ Foreground Color：前景色。

⑥ Background Color：背景色。

选择 Foreground/Background Color 命令调节文本框的颜色，选择 Show Drop Shadow 命令显示文本框阴影，美化 Annotation。

选择 Annotation Properties... Annotation 命令，打开其设置页面，如图 2.2.18 所示。

在 ClikFcn 中输入上文中使用的指令，单击 OK 按钮，完成了 Annotation 的基本设置。在以后重用本模型时，只要单击该文本框就可以执行对模型参数的赋值，如图 2.2.19 所示。

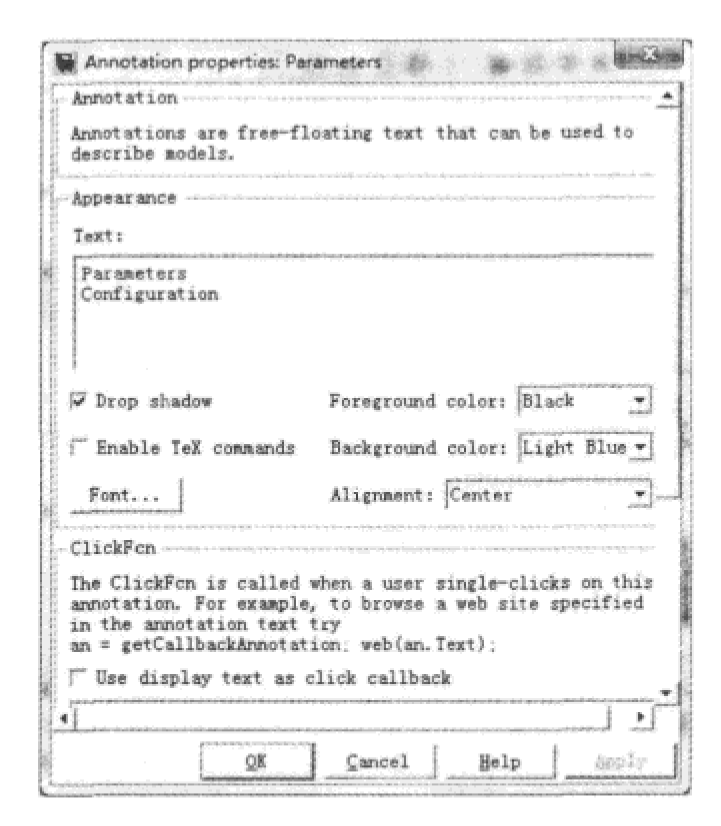


图 2.2.18 设置 Annotation

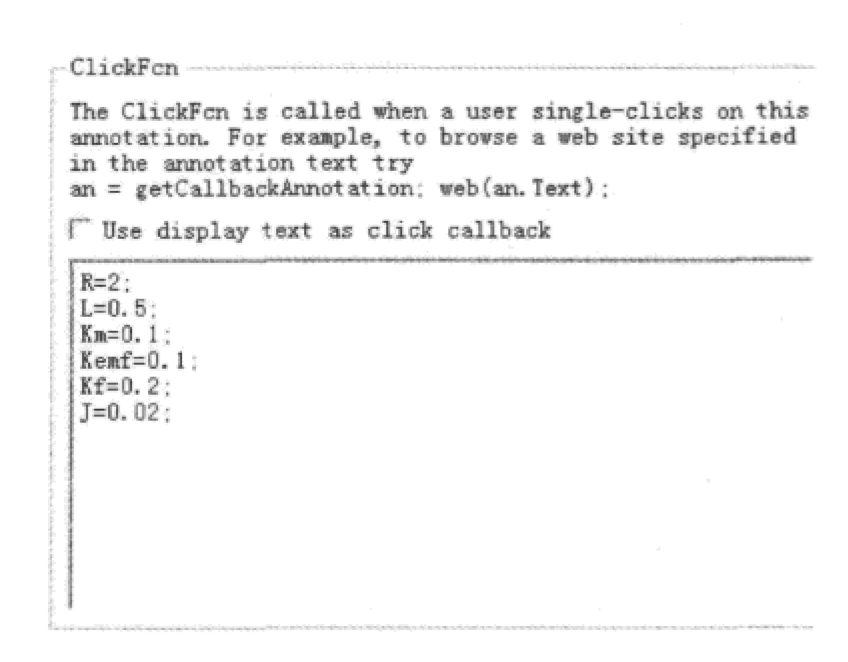


图 2.2.19 添加 Annotation 命令

(3) Model Explorer。Model Explorer 能够快速浏览、批量修改模型中各种元素，而不用再在图形或表格中一个个寻找这些元素。如 Simulink 模块，StateFlow 状态图，MATLAB 工作空间变量等。

在电动机模型编辑窗口右侧单击按钮打开 Model Explorer 界面，如图 2.2.20 所示。

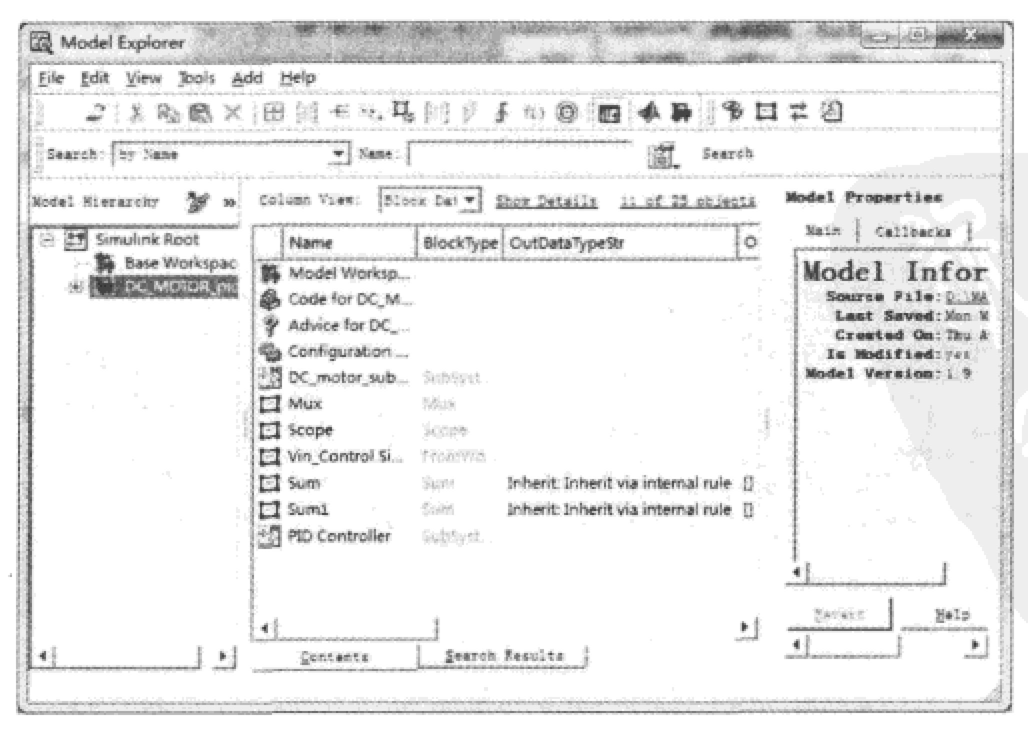
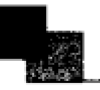


图 2.2.20 模型浏览器



在界面中间的 Content Pane 里列出了模型中的所有模块和变量。选中任意模块可以进行修改参数，数据类型等工作，该模型的所有模块设置都可以在此页面中快速完成。

更详细的用法可参考帮助文档。

(4) 设置 configuration Parameters 中的参数。在 simulation→configuration Parameters 中可进行更多的设置，这里只介绍几种常用的设置。

① 仿真时间设置。设置起止时间是在 Start Time 和 Stop Time 的文本框中输入相应的数值，其默认值分别为 0 和 10，单位是 s，如图 2.2.21 所示。



图 2.2.21　仿真时间设置

② 求解器步长设置。求解器分为变步长和定步长求解器，一般情况下可采用默认的 auto 设置。其默认值为：最大步长＝(停止时间－起始时间)/50

如果是高级用户，则可作进一步设置。步长是求解器运算时的时间精度，步长越短，精度越高，运算量也更大。

通过下面的例子可以直观地看到步长对仿真结果的影响。

建立图 2.2.22 所示的模型。

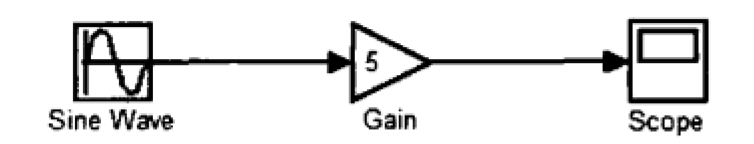


图 2.2.22　求解器步长实验模型

在 Configuration Parameters 中选择使用定步长求解器(Fixed-step)，如图 2.2.23 所示，则当步长(Fixed-step size)分别选择 1 s 和 0.1 s 时的仿真结果如图 2.2.24 所示。

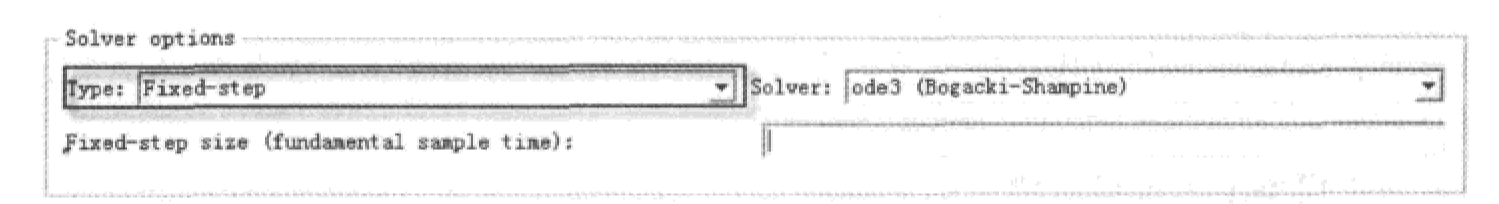
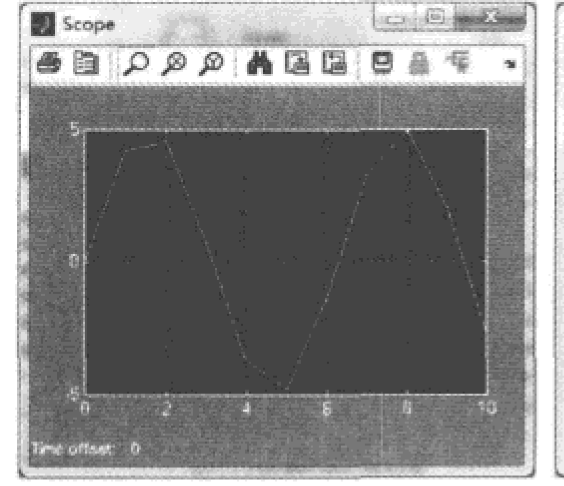


图 2.2.23　求解器设置



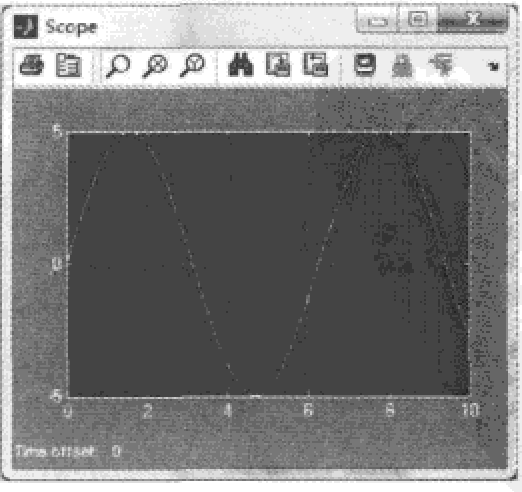
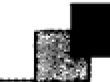


图 2.2.24　仿真结果



可见，当用户对仿真结果的精度要求比较高的时候，应该尽量选择小步长求解。

(5) 运行模型。单击工具栏上的按钮▶运行仿真，双击 Scope 模块即可查看仿真结果（按下按钮🔭可以自动调节坐标轴到合适的范围），如图 2.2.25 所示。

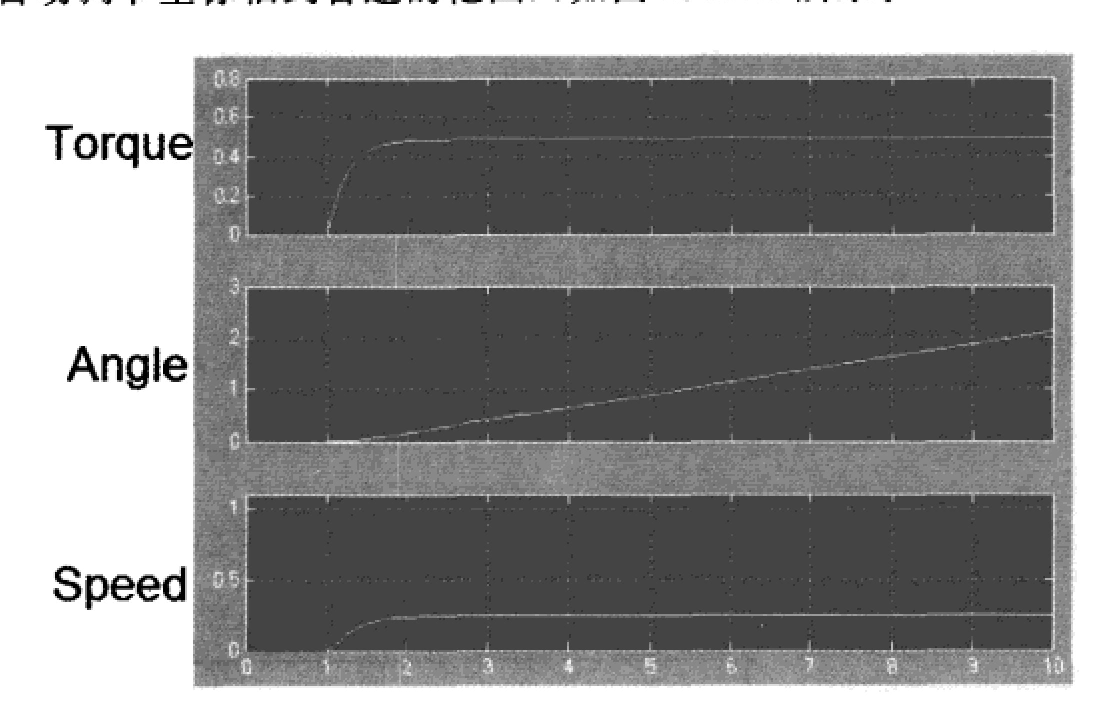


图 2.2.25　电动机模型仿真结果

在示波器 Torque 中可见转矩跟随阶跃信号的趋势变化。

在示波器 Angle 中可见，转子转动过的角度随时间变化基本呈线性增长，符合其与 $w(t)$ 的关系 $\theta=\int_0^t w(\tau)\mathrm{d}\tau$。

在示波器 Speed 中，紫色信号表示阶跃信号；黄色信号为系统响应，在接近 2 s 的时候达到稳定状态，幅度为 0.25 左右。

为了更进一步分析此系统，可以在 MATLAB 中画出其波特图：

```
>> DC_motor = tf([Km],[J * L (Kf * L + J * R) (Kf * R + Kemf * Km)]);     % 生成电动机传输函数
>> bode(DC_motor)                                                           % 生成波特图
```

在 MATLAB 中生成图 2.2.26 所示波特图。

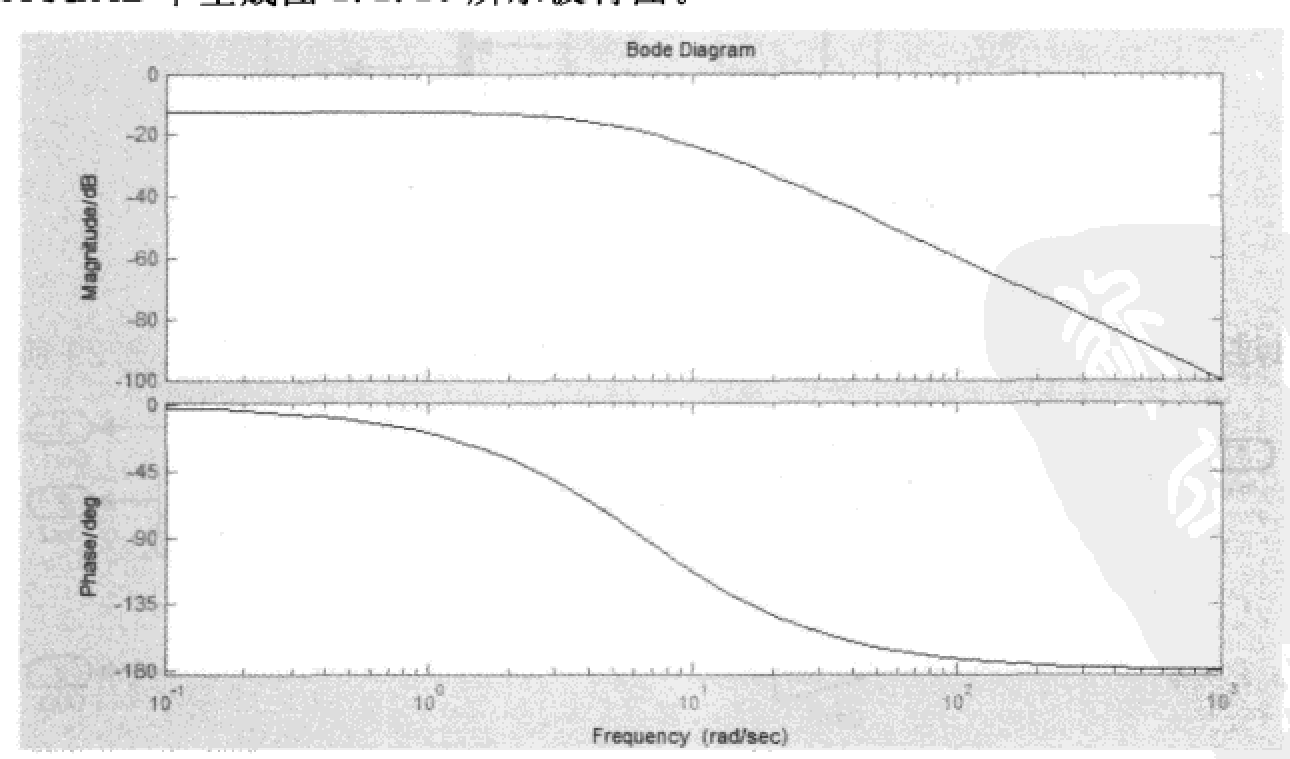


图 2.2.26　波特图

2.2.3 子系统与库

建立子系统可以把一系列的模块表示为一个子系统模块,大大简化模型的复杂度,其具有以下突出优点:减少编辑窗口中模块的数量;可以把功能相关联的模块组合在一起;使模型具有明确的功能层次。

更重要的是在子系统内部,各个模块被认为是一个整体,这样一来,其运算速度和信号传递速度都会大幅度提高,是对模型的一种优化。

1. 建立子系统

如果模型中已经完全包含了你想要创建的子系统的所有模块,可以通过把它们组合起来的方式建立子系统。

打开电动机模型,选定想要包含到子系统中的所有模块,如图 2.2.27 所示。

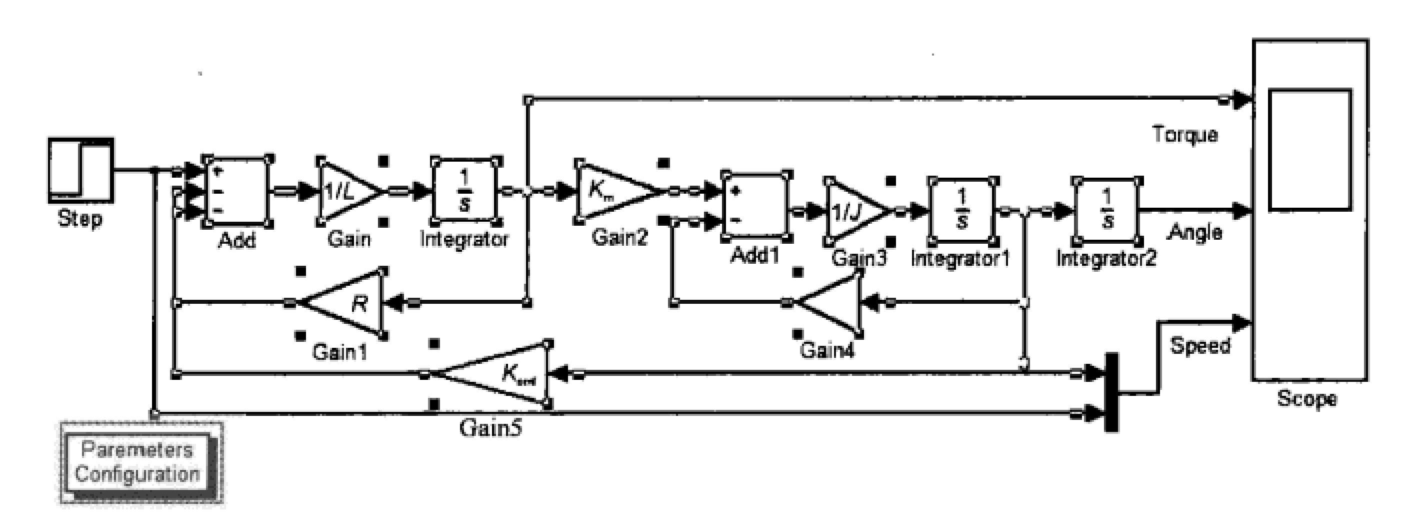


图 2.2.27 选定模块

单击 Edit→Creat Subsystem 命令。选定的模块组即变成了一个子系统模块,如图 2.2.28 所示。

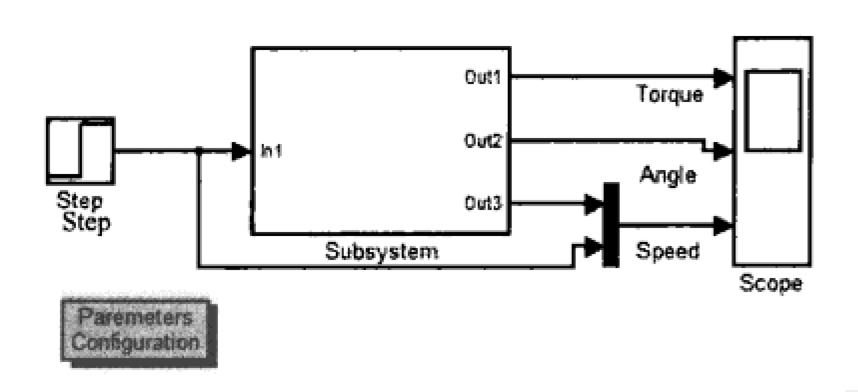


图 2.2.28 创建子系统

如果通过双击打开子系统,可以看到子系统中的模块组合逻辑,如图 2.2.29 所示。

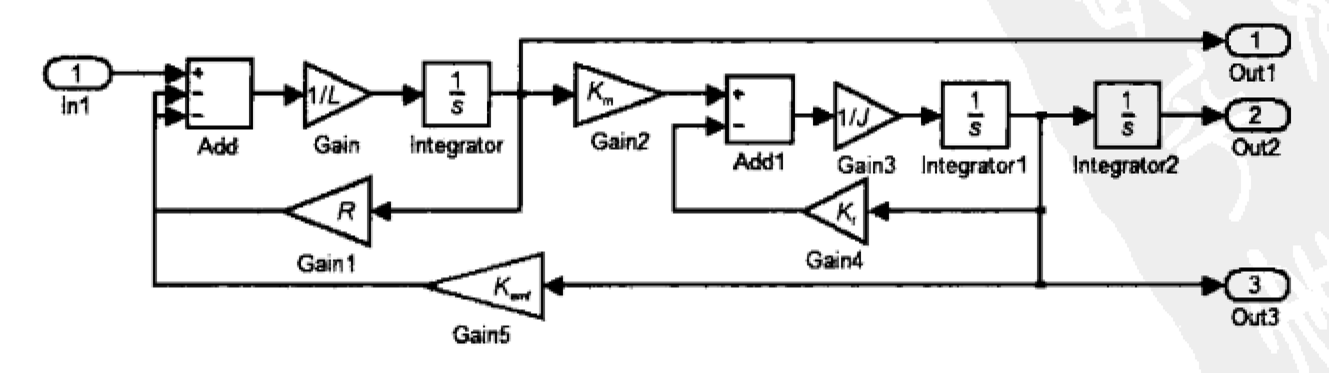
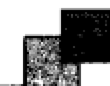


图 2.2.29 子系统内部结构



子系统外部接口也会发生相应的改变，可按图 2.2.30 所示方式连接模型。

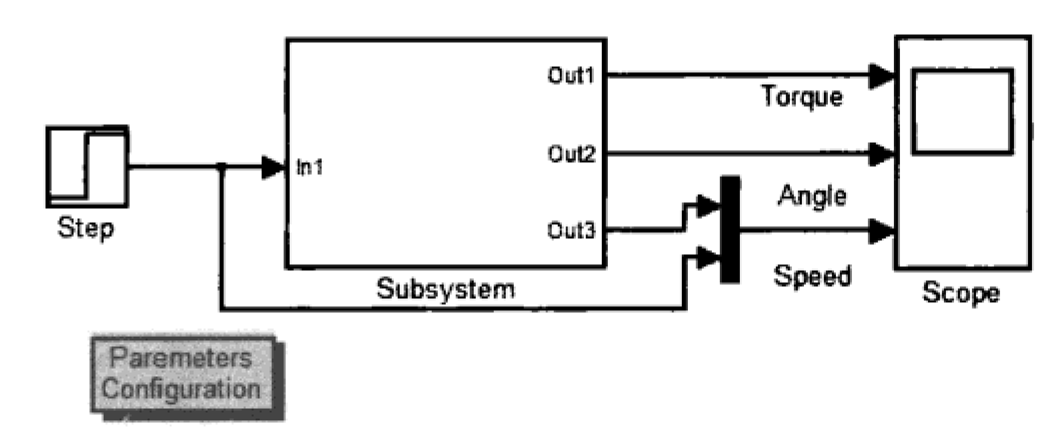


图 2.2.30　修改连接

用户也可以先创建一个子系统模块，然后在子系统模块中完成其具体的组合逻辑。

从模块浏览器中选择 Ports&Subsystems library→Subsystem。通过双击打开子系统模块。在子系统中完成其组合逻辑，如图 2.2.31 所示。

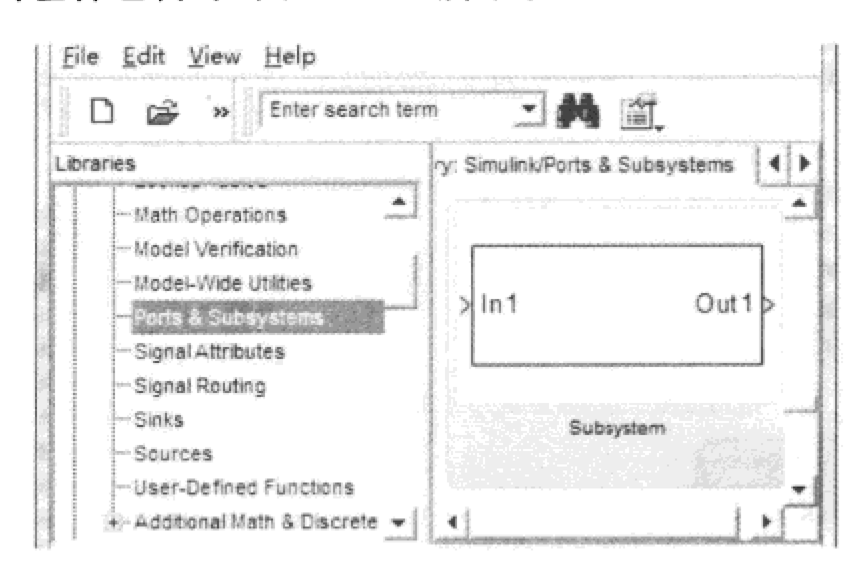


图 2.2.31　Subsystem 模块

2. 封装子系统

将子系统模块重命名为 DC_motor_subsystem，方便以后使用。右击子系统模块，在弹出的右键菜单中选择 Mask Subsystem，打开封装编辑器。

(1) Icon & Ports 选项卡。在该页面的 Icon options 区域，可以设置模块外框是否可见，模块是否透明，旋转时图标是否固定或跟随模块旋转，以及图标是否自适应模块的大小，这里均采用默认设置。

Icon Drawing Commands 区域中可以用命令改变模块的端口名称，添加图像，修改颜色等，如图 2.2.32 所示。

按照界面下方的 Example Drawing Commands 给出的语法格式，为 DC_motor_subsystem 添加如下命令：

```
color('red');port_label('input', 1, 'Control Signal');
color('red');port_label('output', 1, 'Torque');
color('red');port_label('output', 2, 'Angle');
color('red');port_label('output', 3, 'Step');
color('red');port_label('output', 4, 'Speed');
```

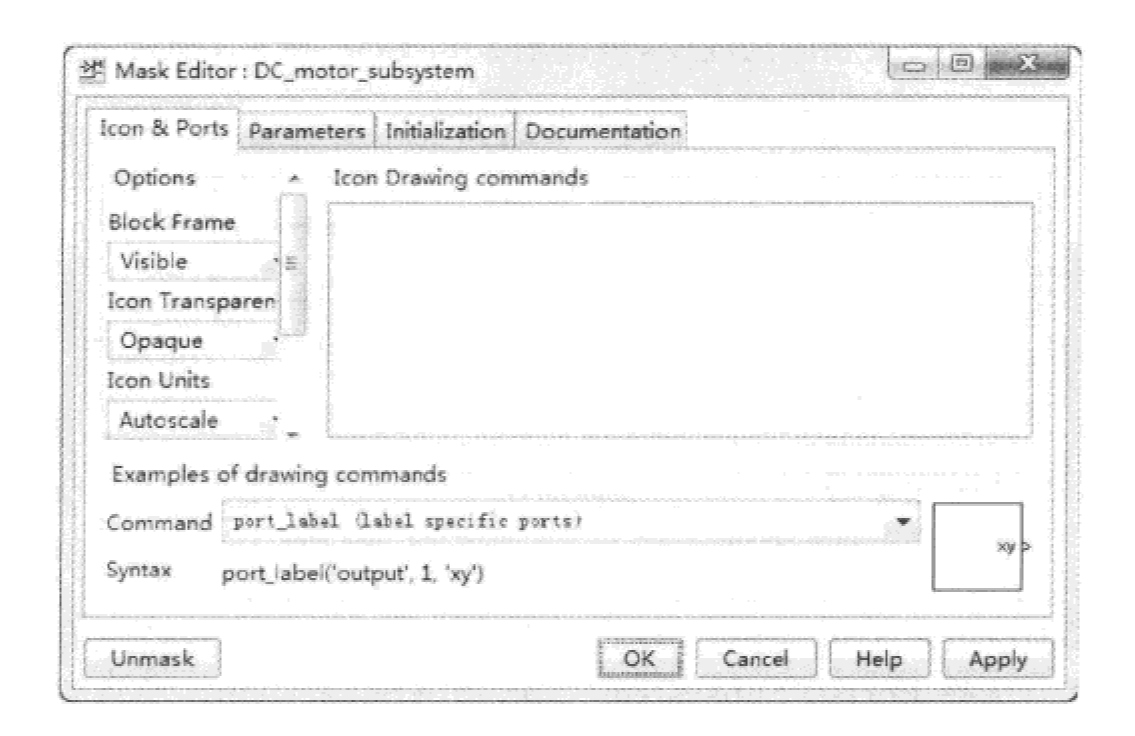


图 2.2.32　Icon&Ports 选项卡

子系统的端口名称的颜色和名称都会随之改变，如图 2.2.33 所示。

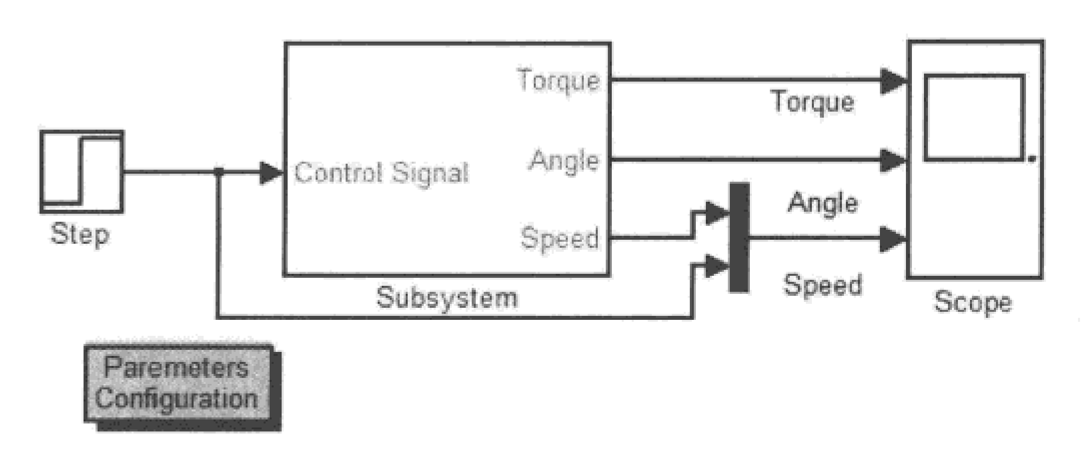


图 2.2.33　封装后的模型

添加命令 image(imread('dianji.jpg'))；

会将当前工作目录下的 dianji.jpg 图像文件覆盖到模块上，非常直观地说明该子模块的作用如图 2.2.34 所示。

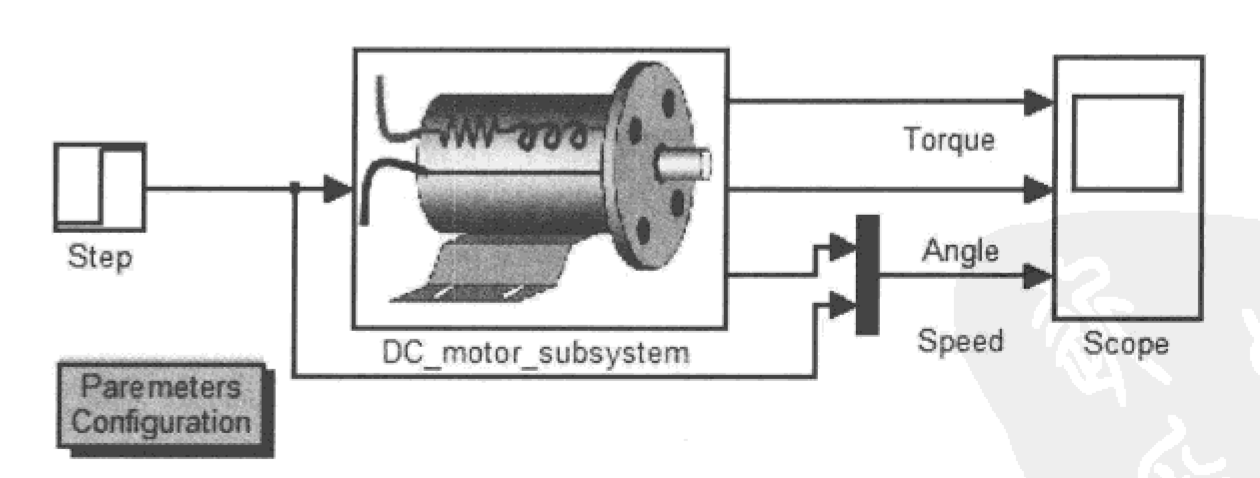
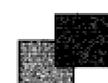


图 2.2.34　封装后的模型

单击 Example Drawing Commands 的下拉菜单可以看到所有的绘图指令和语法格式，用户可根据需要添加相应的命令。

(2) Parameters 选项卡。单击左侧工具栏按钮 ，Dialog parameters 区域新增一个参数



设置框，各栏的功能如下：

① Prompt：参数的文字描述。

② Variable：参数对应的变量名。

③ Type：参数的输入模式，edit 适用于需要用户具体指定的参数。

④ Evaluate：选中表示在参数对话框中输入表达式的值赋予指定的变量，否则将输入的表达式看作字符串赋予指定的变量。

⑤ Tunable：选中则允许用户在仿真过程中修改该参数值。

在本例中，按图 2.2.35 所示的方案设定参数。

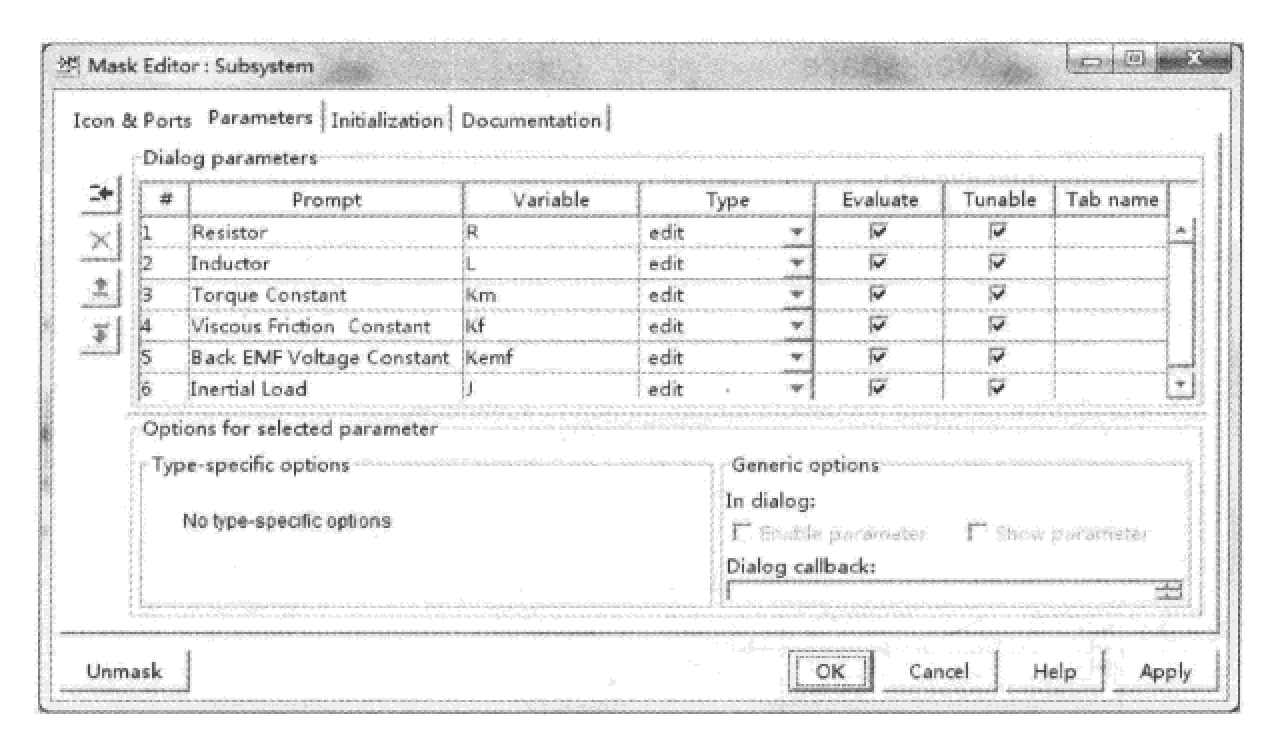


图 2.2.35　Mask 参数设置界面

单击 OK 按钮确认后，若双击子系统模块，将不会再显示子系统底层结构，而会像 Simulink 模块库中自带的模块一样弹出参数设置界面，如图 2.2.36 所示。

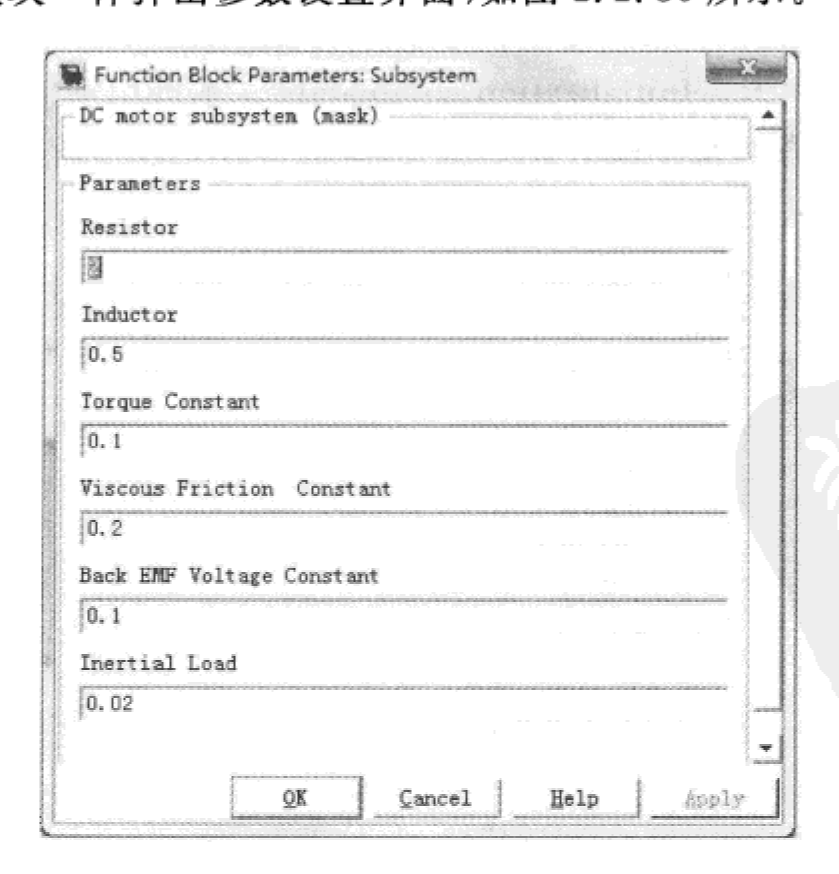


图 2.2.36　封装后的参数设置界面

封装后的系统参数其实是与底层模块的参数相关联的，给封装系统的参数赋值就相当于给定义于封装工作空间中的参数赋值，而这些参数与底层的一个或多个模块相关。以电阻值 R 和力矩常量 K_m 为例说明封装参数与底层模块参数间的关系，如图 2.2.37 所示。

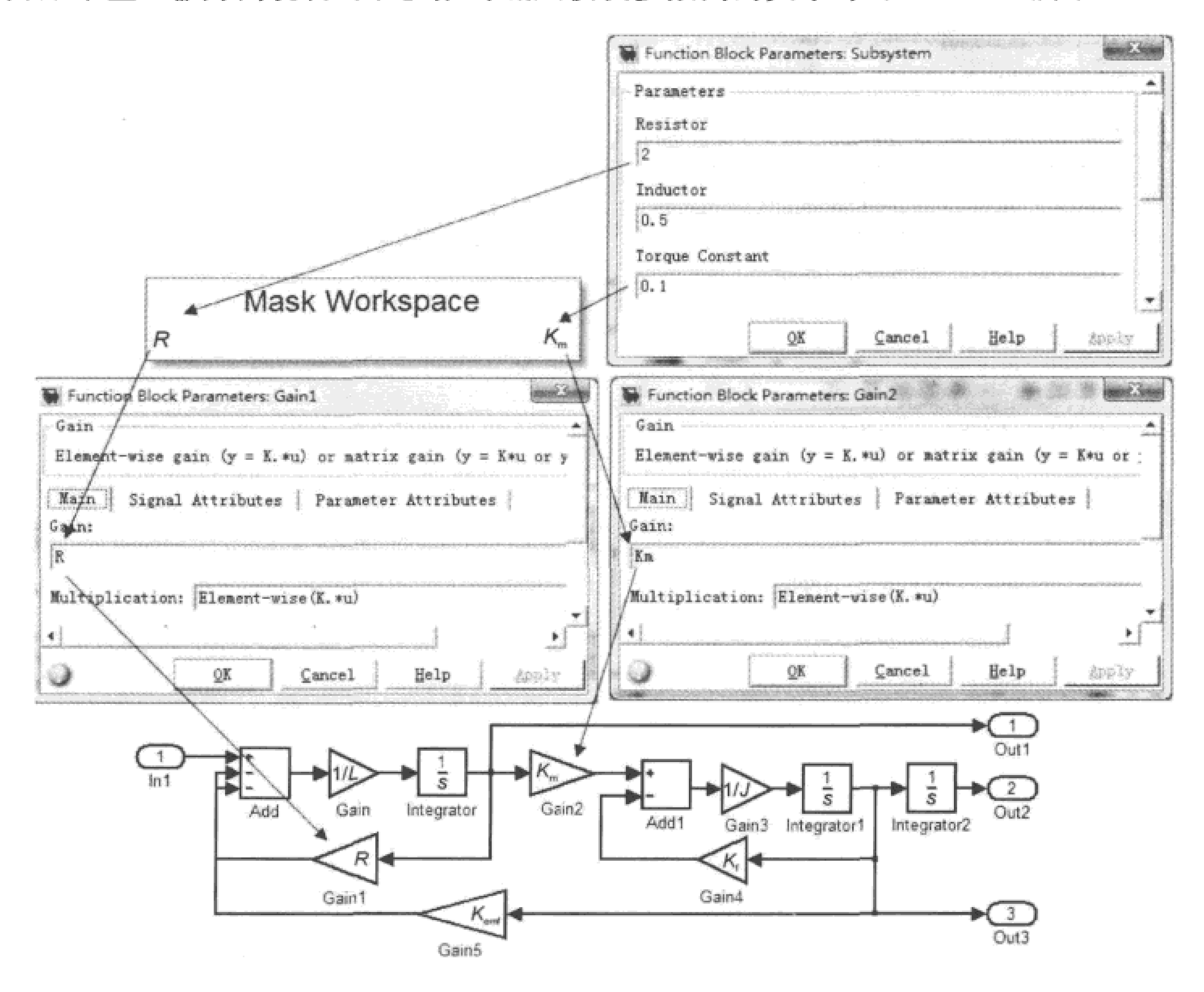


图 2.2.37　参数传递关系

(3) Initialization 选项卡。Initialization commands 文本框用于输入用于初始化该模块的 MATLAB 命令。如果在该界面中输入如下命令，同样可以完成参数的赋值工作，如图 2.2.38 所示。

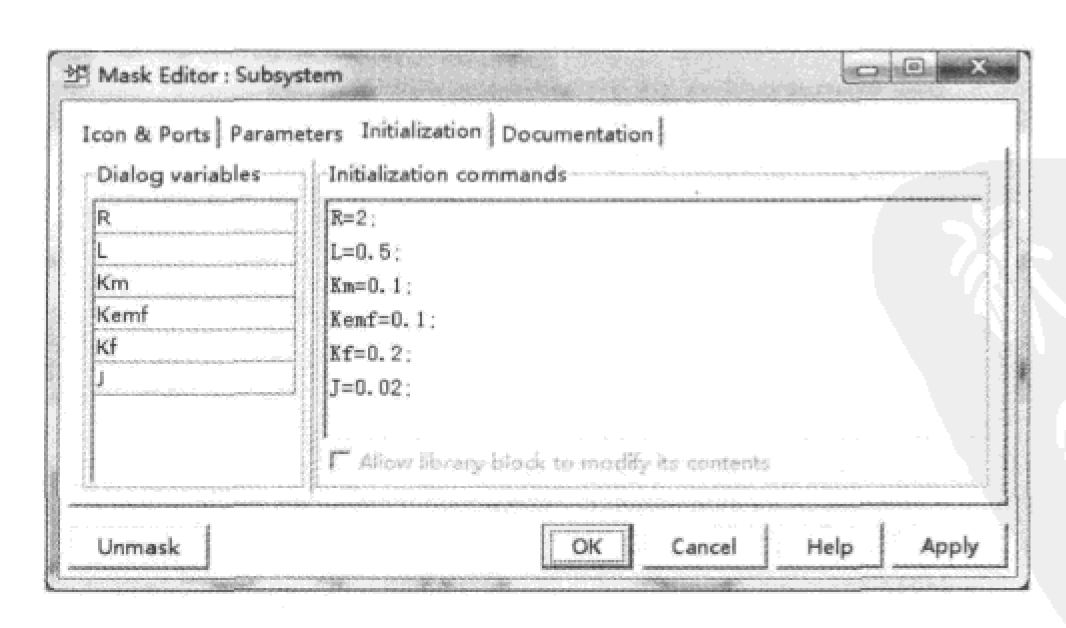
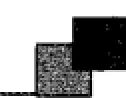


图 2.2.38　Initialization 选项卡



但是这种方式应慎用，这会导致上文中所讲的Annotation、M文件等赋值方法统统失效，这是由于最终在模块执行之前它都会执行此初始化命令，将参数的值重新定义为初始化页面中设置的量。

(4) Documentation页面。在此页面中可以注释封装的类型、描述、帮助等信息。

① Mask type：文本框可写入对模块的简单描述。

② Mask description：文本框用于详细描述模块的功能。

③ Mask help：文本框用于指定模块的帮助文档。

用户可根据需要编辑相关信息，如图2.2.39所示。

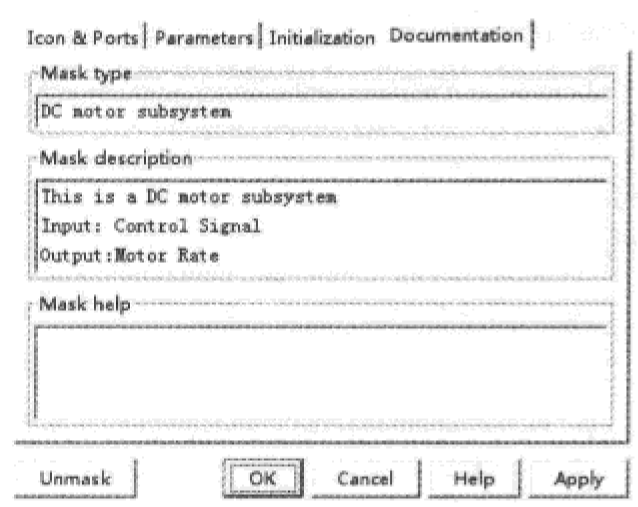


图2.2.39 Documentation选项卡

2.2.4 添加模块到库浏览器及知识产权保护

在设计大型模型时，通常会遇到多次重复使用一个子系统模块或团队的其他成员需要使用用户所搭建的子系统的情况，用户可以建立自定义的模块库并添加到模块浏览器中，将这些子系统归类整理，分门别类地管理，便于后期维护与利用。

1. 建立自定义模块库并添加到库浏览器

(1) 建立自定义模块库。在Simulink库浏览器窗口，选择菜单项File→New→Library，打开库编辑窗口。打开电动机模型，将封装后的子系统拖入库编辑窗口，并保存为DC_motor_subsystem_lib。如图2.2.40所示。

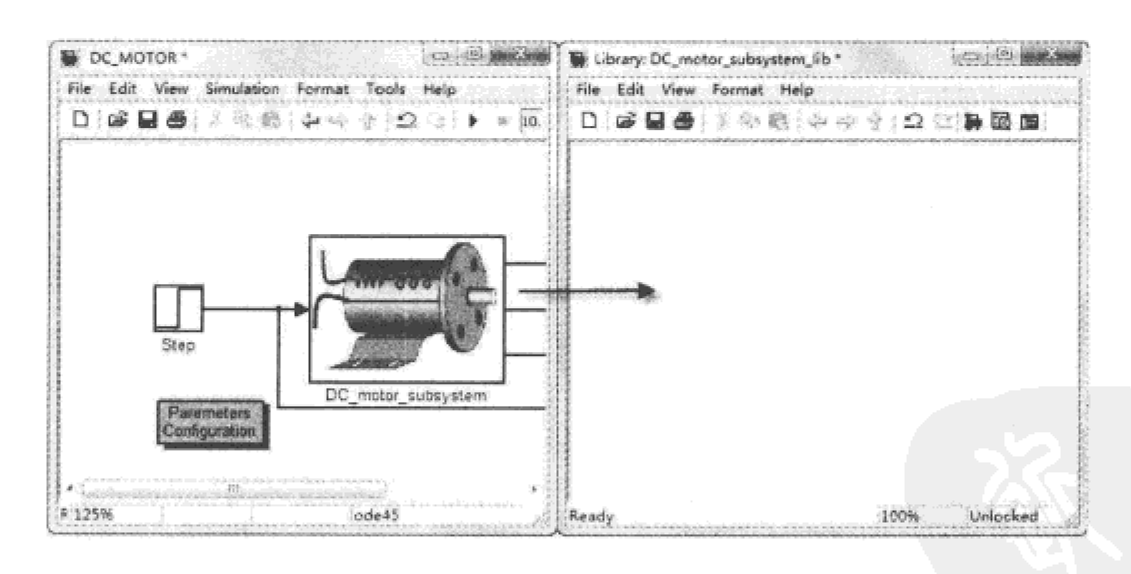


图2.2.40 建立自定义模块库

(2) 将自定义模块库添加到库浏览器中。将自定义模块库添加到库浏览器中，可以在以后的工作中更方便快捷地调用。MATLAB中已经为用户提供了添加自定义模块库的相关函数，用户可以编写以下M函数代码：

```
function blkStruct = slblocks
blkStruct.Name = ['DC_motor_subsystem_lib'];
```

```
blkStruct.OpenFcn = 'DC_motor_subsystem_lib';
Browser(1).Library = 'simulink'; Browser(1).Name = 'Simulink';
Browser(1).IsFlat = 0;
Browser(2).Library = 'DC_motor_subsystem_lib';
Browser(2).Name = 'DC_motor_subsystem_lib';
Browser(2).IsFlat = 0;
```

将其保存为 slblock.m 并保存到 DC_motor_subsystem_lib.mdl 所在的目录下,如图 2.2.41 所示。打开模块库浏览器,自定义的模块库已经成功添加。该模块可以直接使用,如图 2.2.42 所示。

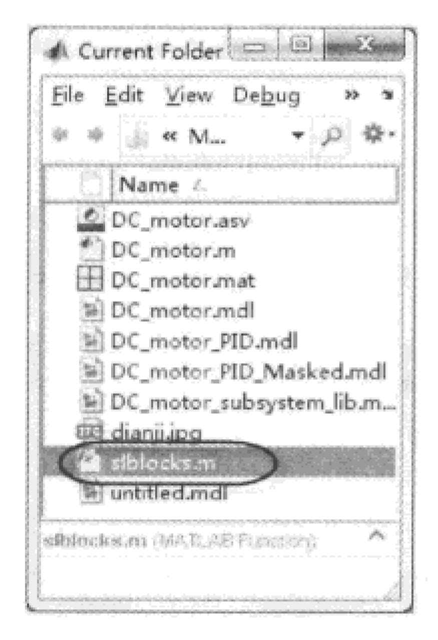


图 2.2.41 M 文件存储路径

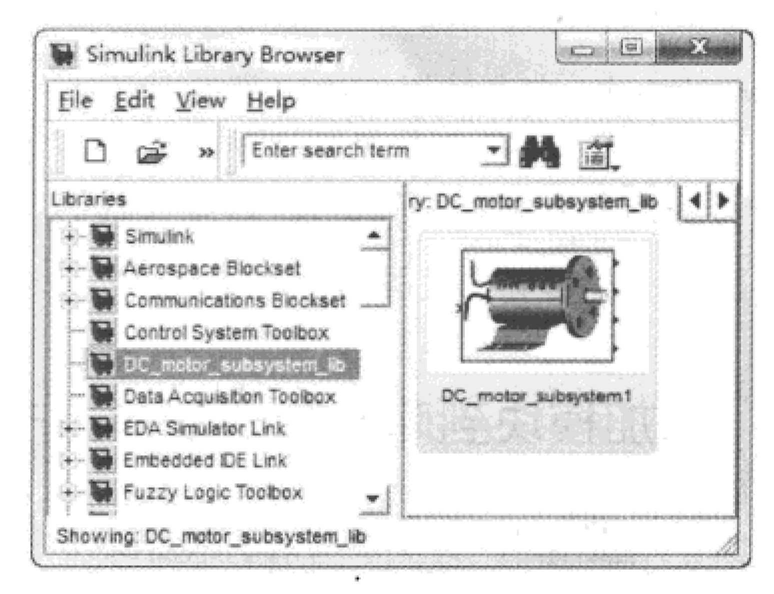


图 2.2.42 添加到模块库浏览器

2. 知识产权保护

有时用户需要把自定义的模块库提供给别人使用,但是却不想让别人看到或修改子系统的底层模块结构。通过以下设置可以在不影响使用的条件下禁止他人访问底层模块,有效保护了用户的个人资料和知识产权。

(1) 在将封装后的子系统拖入库编辑窗口之前右击子系统,选择 subsystem parameters 命令,如图 2.2.43 所示。

(2) 在 Read/Write Permissions 下拉菜单中选择 NoReadOrWrite 命令,单击 OK 确认。

(3) 将子系统模块拖入库编辑窗口。

在库编辑窗口中右击模块,可以发现这时 Look Under Mask 选项变成了灰色,禁止访问底层模块,如图 2.2.44 所示。

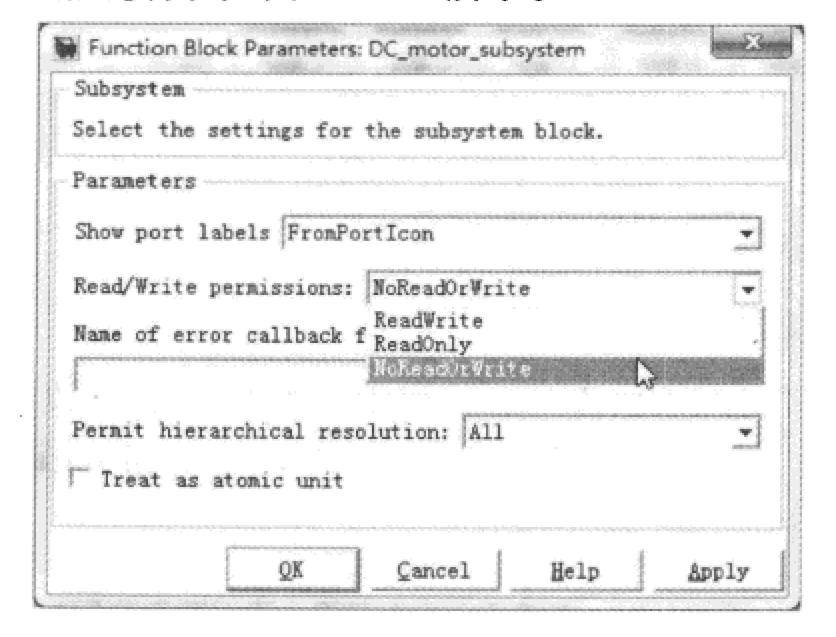


图 2.2.43 Block Parameters 界面

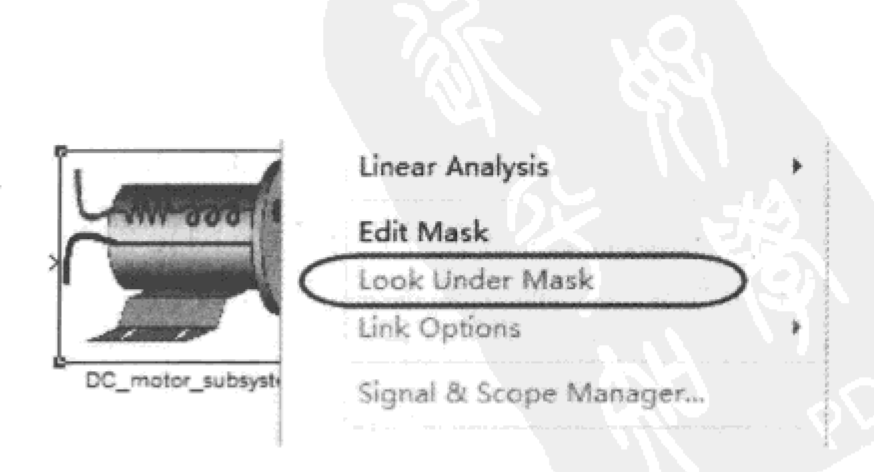
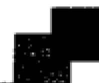


图 2.2.44 保护后的模块



2.2.5　数据格式与输入/输出

Sources 或 Sinks 子库中的现成信号源或输出模块可能不能满足实际应用，因此用户可以根据需要选用 From Workspace 及 To Workspace 模块，自行定义模型的输入及输出。

用户可以通过上述模块导入 MATLAB 工作空间的数据，也可以把输出信号保存到工作空间。这项功让用户能够使用由标准或自定义的 MATLAB 函数产生的数据作为输入信号和图形，使仿真更具多样性。

1. 把数据导出至工作空间

(1) 新建一个演示模型，向模型中添加 sink 库中的 To Workspace 模块，该模块位于图 2.2.45所示的位置。

打开 To Workspace 模块的参数设置界面，其中 Data 栏显示的 simout 即变量名，用户可根据需要自行更改。

为便于 From Workspace 访问，可以将 To Workspace 模块的 save format 设置为 Array，如图 2.2.46 所示。

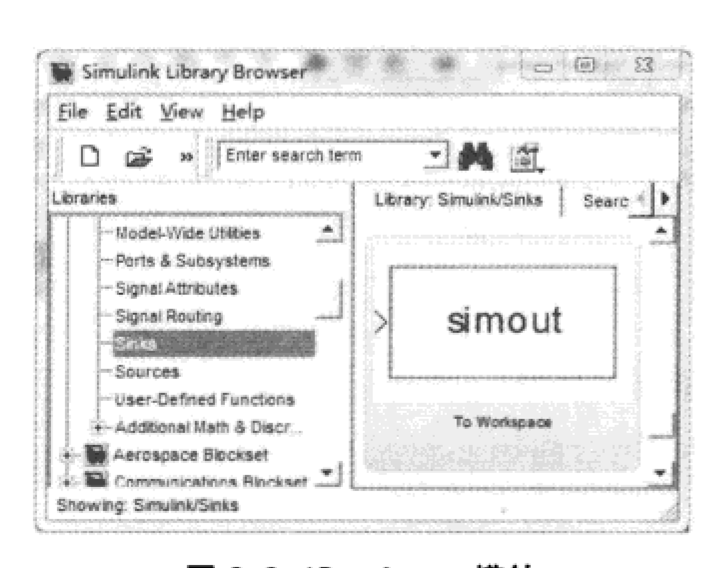


图 2.2.45　simout 模块

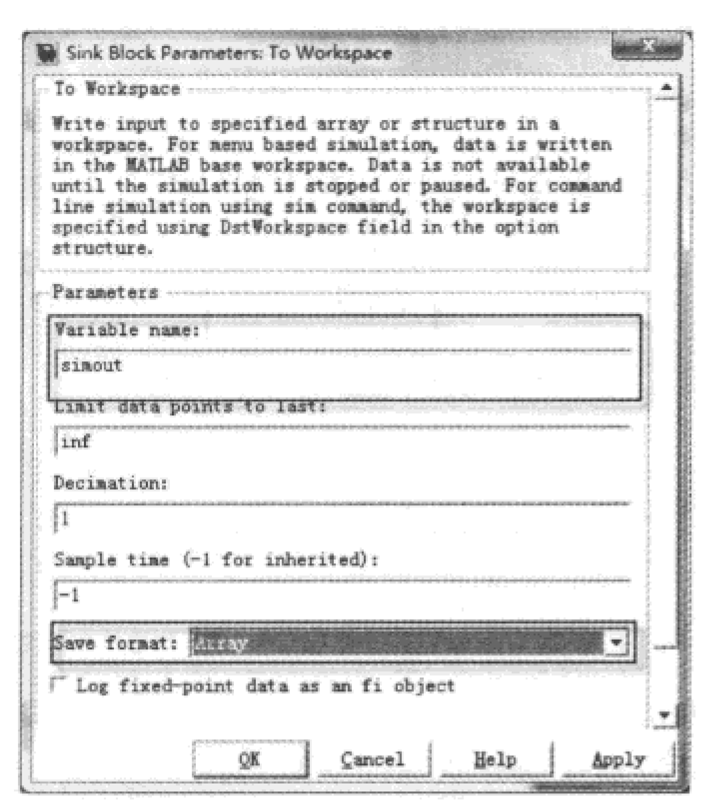


图 2.2.46　设置 simout 模块

(2) 向模型中添加 source 库中的 signal builder 模块，如图 2.2.47 所示。

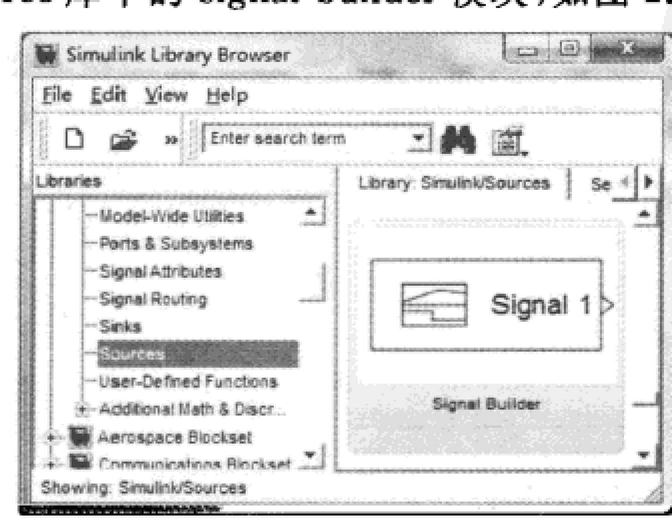


图 2.2.47　signal builder 模块

打开该模块的设置界面，用户可以方便地通过其 GUI 界面画出任意需要的信号波形。这里画出一个从第 3 s 起持续 4 s 的矩形脉冲，如图 2.2.48 所示。关于该模块的更详细用法不是本书重点，如有兴趣可参看帮助文档。

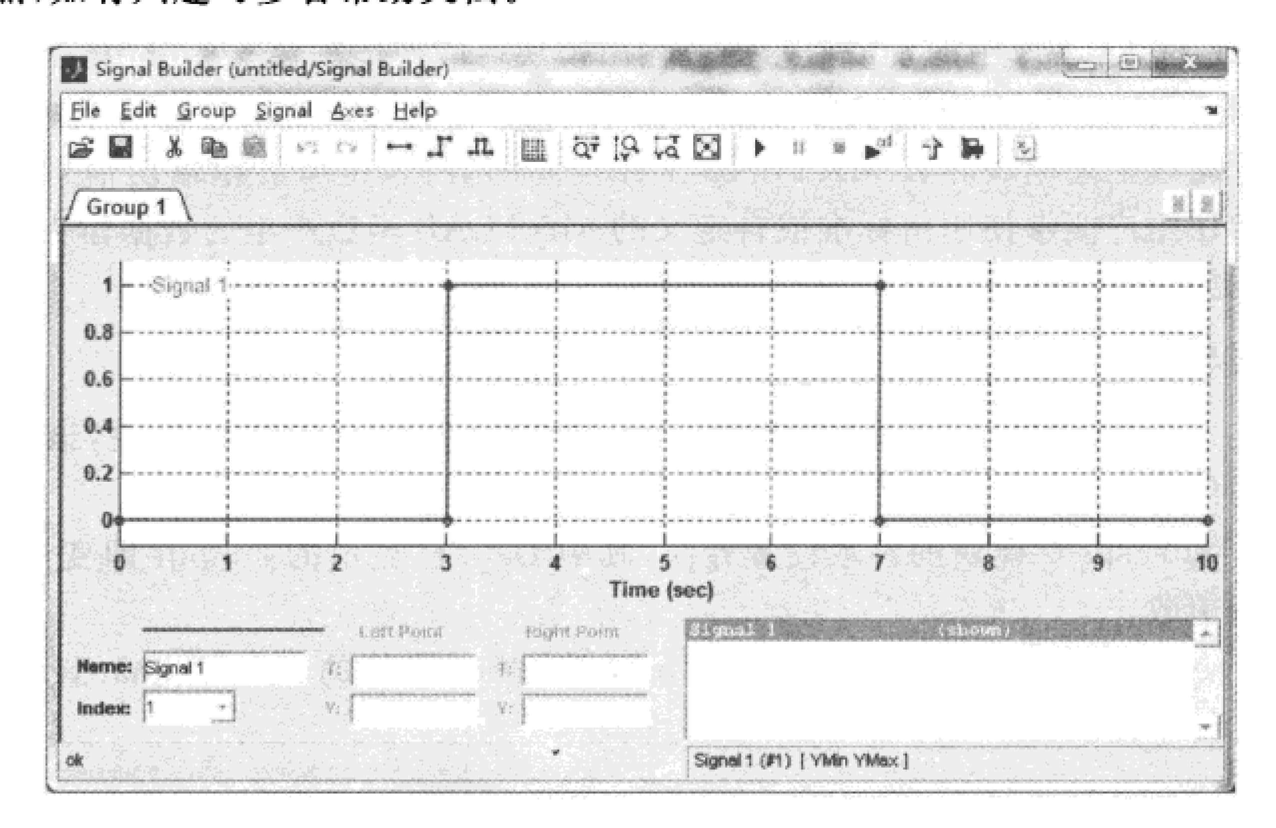


图 2.2.48 信号波形

(3) 向模型中添加 gain 模块和 scope 模块，并按图 2.2.49 连接模块。

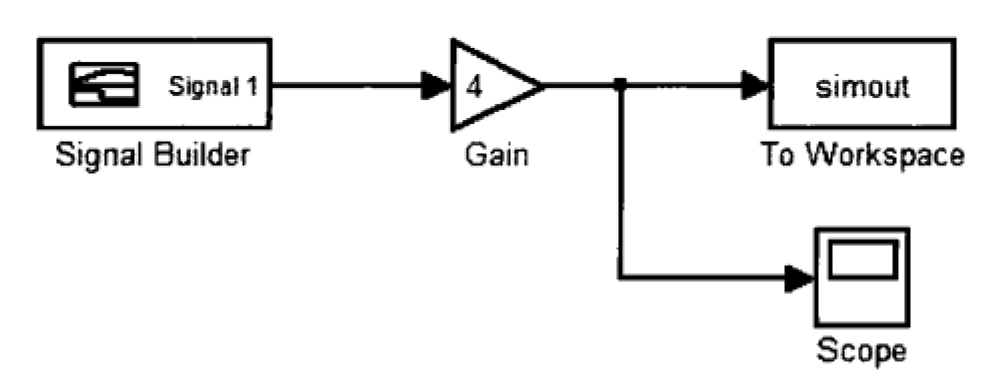


图 2.2.49 向工作空间传递数据

单击工具栏上的按钮▸运行仿真，MATLAB 的工作空间中会显示变量如图 2.2.50 所示。示波器中显示波形如图 2.2.51 所示。

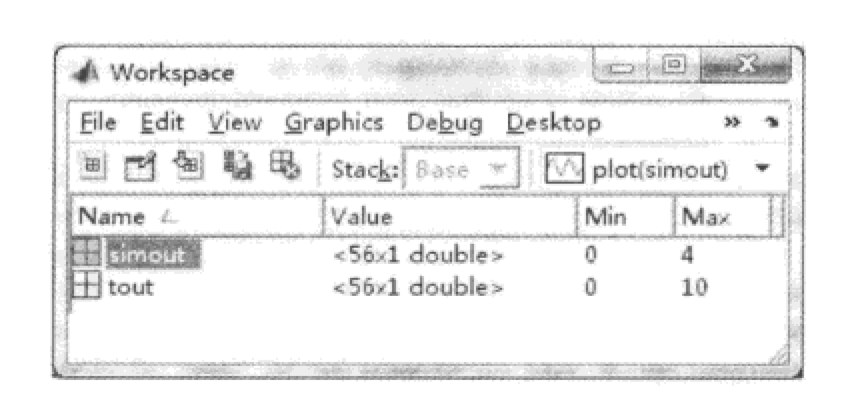


图 2.2.50 工作空间的数据

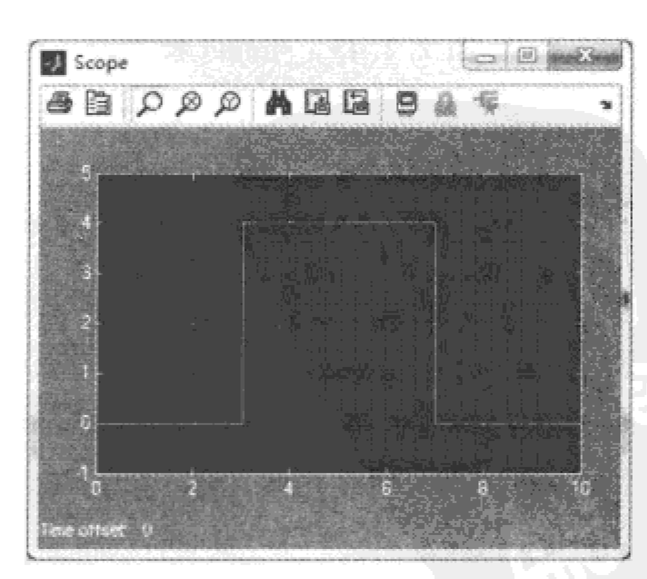
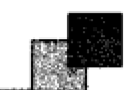


图 2.2.51 仿真波形



Simulink 仿真中的数据已经导入了工作空间，用户可以根据需要对这些数据作进一步处理。

2. 从工作空间导入数据

(1) 向模型中添加 sources 库中的 From Workspace 模块代替 signal builder，即可用于从工作空间导入数据，该模块位于图 2.2.52 所示的位置。

打开 From Workspace 模块的参数设置界面，其中 Data 栏显示的 simin 即变量名，用户可根据需要自行更改，如图 2.2.53 所示。

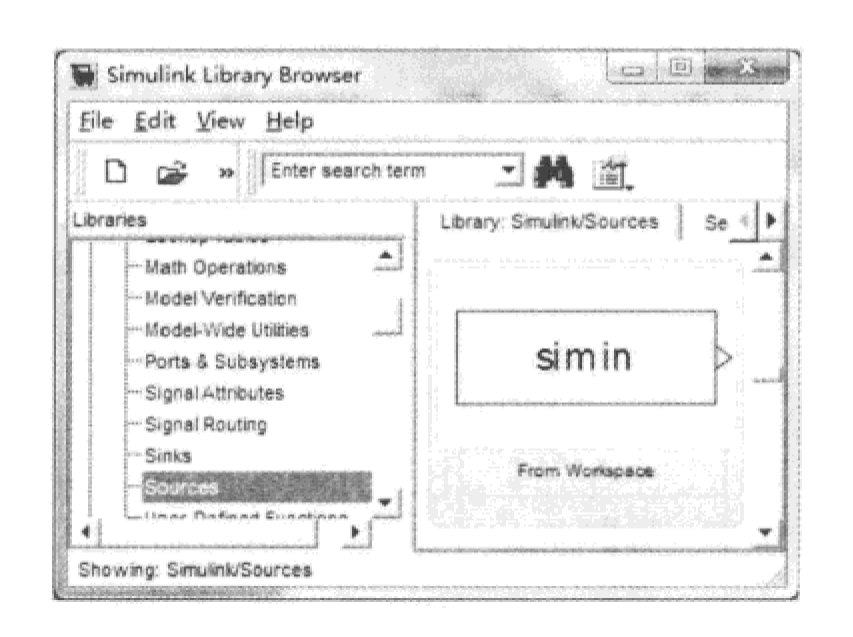


图 2.2.52 simin 模块

Source Block Parameters: From Workspace

From Workspace

Read data values specified in matrix or structure format f
MATLAB's workspace.

Matrix format can be used only for one-dimensional signals
Each row of the matrix has a time stamp in the first colum
a vector containing the corresponding data sample in the
subsequent column(s).

Structure format can be used for either one-dimensional or
multidimensional signals:
var.time=[TimeValues]
var.signals.values=[DataValues]
var.signals.dimensions=[DimValues]

Parameters

Data:

simin

Sample time:

0

OK Cancel Help

图 2.2.53 simin 设置

(2) 按图 2.2.54 所示连接模块。

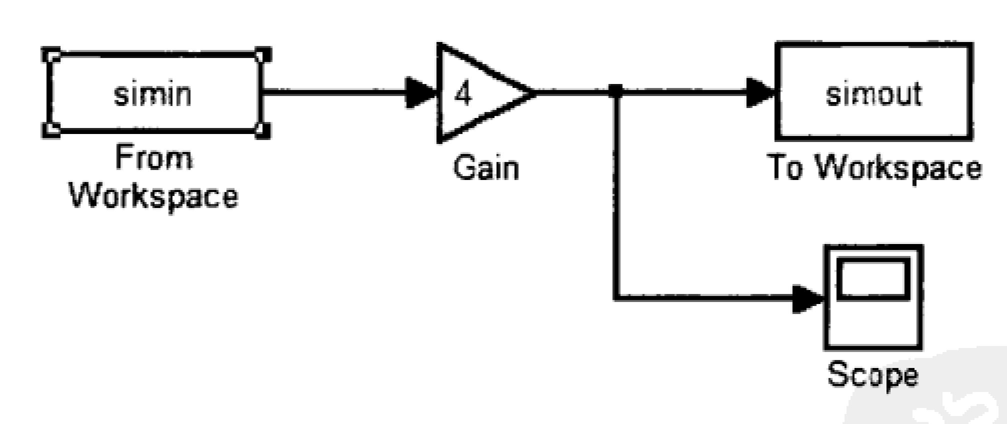
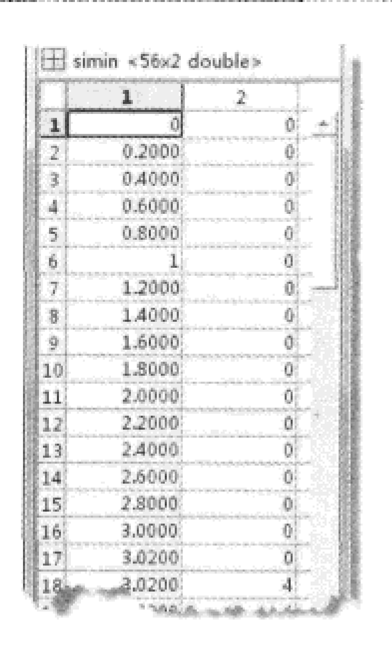


图 2.2.54 从工作空间读数据

From Workspace 模块调用工作空间的数据时，是遵循严格的格式的，一般是矩阵形式，而且输入矩阵的列数必须比输入信号多一列，Simulink 会自动把首列数据当做时间向量。根据此规则，需要在工作空间内生成一个待输入信号：

```
>> simin = [tout simout];生成 2 列的矩阵 %
```

用户可以打开 workspace 中的变量 simin 查看其数据存储结构，如图 2.2.55 所示。



simin <56x2 double>

	1	2
1	0	0
2	0.2000	0
3	0.4000	0
4	0.6000	0
5	0.8000	0
6	1	0
7	1.2000	0
8	1.4000	0
9	1.6000	0
10	1.8000	0
11	2.0000	0
12	2.2000	0
13	2.4000	0
14	2.6000	0
15	2.8000	0
16	3.0000	0
17	3.0200	0
18	3.0200	4

图 2.2.55　数据存储结构

3. 实现外部信号对电动机的控制

回到电动机模型，用一个外部信号替代阶跃信号控制电动机，用户可以通过这个外部信号实现对电机转速的控制。

向模型中添加 From Workspace 模块替代 Step 模块，并重命名为 Vin_Control Signal，如图 2.2.56 所示。

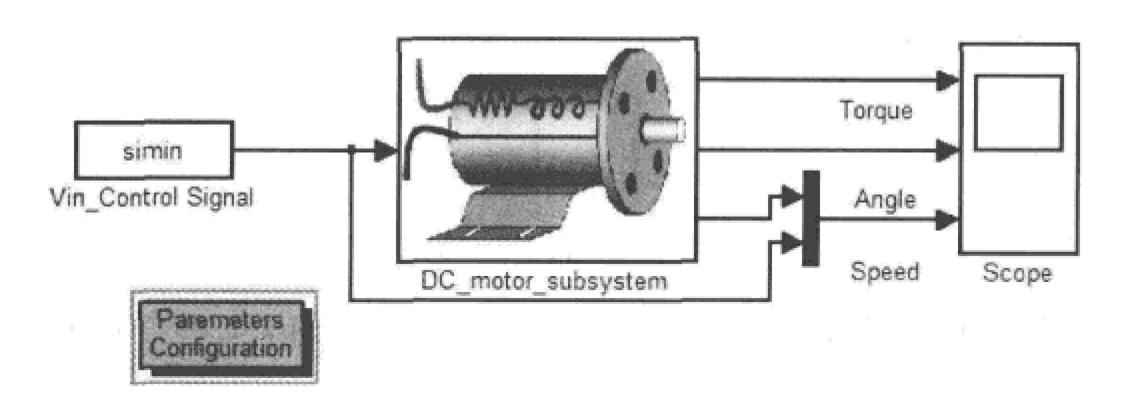
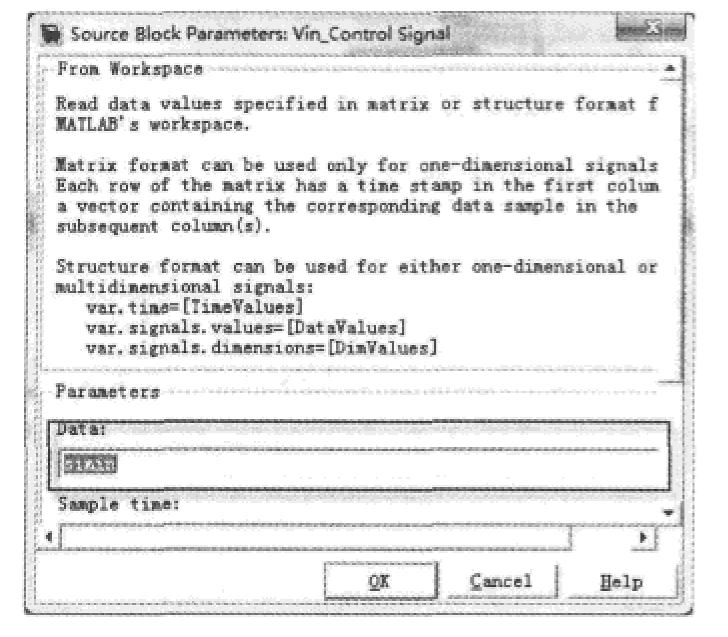


图 2.2.56　添加 simin 模块

这里的 From Workspace 模块仍然使用上文中生成的变量 simin 作为控制信号，Data 栏设置变量名为 simin 即变量名，如图 2.2.57 所示。

运行仿真后得到的结果如图 2.2.58 所示。



Source Block Parameters: Vin_Control Signal

From Workspace

Read data values specified in matrix or structure format f MATLAB's workspace.

Matrix format can be used only for one-dimensional signals Each row of the matrix has a time stamp in the first colum a vector containing the corresponding data sample in the subsequent column(s).

Structure format can be used for either one-dimensional or multidimensional signals:
var.time=[TimeValues]
var.signals.values=[DataValues]
var.signals.dimensions=[DimValues]

Parameters

Data:

simin

Sample time:

OK　Cancel　Help

图 2.2.57　设置 simin 模块

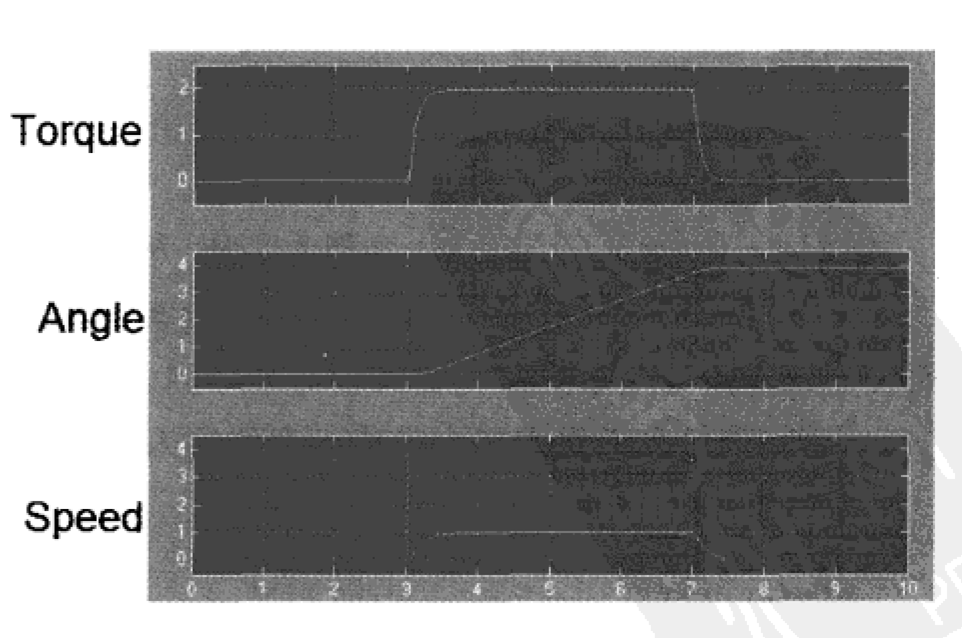
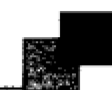


图 2.2.58　仿真结果



由仿真结果可知,系统响应曲线变化的总体趋势是跟随控制信号相吻合的,说明外部信号对电动机转速的控制是有一定效果的。但系统响应速度还不理想,且其响应幅度远远没有达到控制信号的要求,需要通过进一步矫正才能符合需要。

2.2.6 PID 控制

在上文的仿真结果中可以看到:输出电动机转速 $\omega(t)$ 只是大体上跟随控制信号的变化趋势,而并没有严格地达到控制信号所要求的转速大小。这是由于模型还缺少一个反馈控制模块,PID 模块能很好的完成这个工作。

PID(比例积分微分)英文全称为 Proportion Integration Differentiation,它以结构简单、稳定性好、工作可靠、调整方便等特点成为工业控制的主要技术之一。

PID 控制器由比例部分,积分部分和微分部分构成(图 2.2.59)。控制器的输入端是被控系统的控制信号与被控量间的误差,误差信号分别通过比例、积分、微分环节,并与相应的系数相乘,输出的 PID 控制器信号即为这 3 个部分输出信号之和,最终 PID 控制器输出信号与原始控制信号做差后控制被控量,最终形成完整的反馈回路。因此,要确定一个 PID 控制器只需要确定其比例、积分、微分的系数:K_p、K_i、K_d,如图 2.2.59 所示。

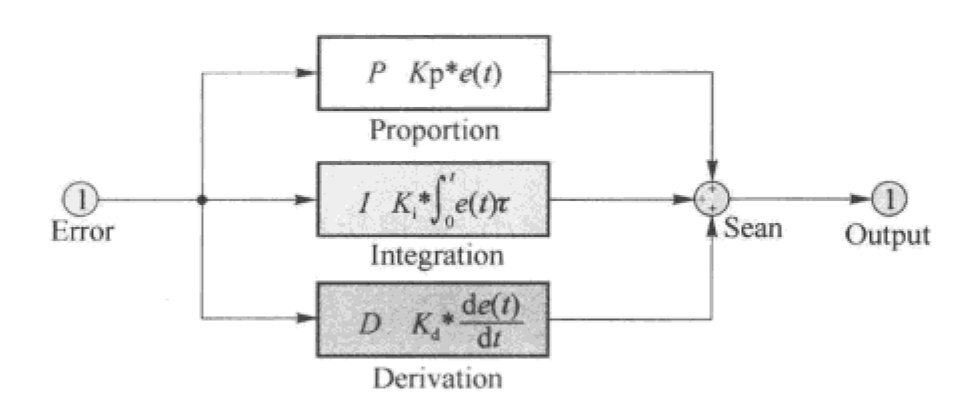


图 2.2.59 PID 控制器结构

PID 控制器的参数设定是控制系设计的核心内容,但是传统的设计方法存在不少缺陷,在实际应用中会遇到很多问题:

(1) 传统的设计方法中,参数 K_p、K_i、K_d 的设定主要依赖以往的工程经验,对控制器的参数进行手动调节,如果以前的项目中有过类似的控制器,就可以直接在控制系统硬件上进行手动微调,这样工作量很小。但如果被控对象是一个不稳定系统,手动调节就会变得比较复杂,因为很可能由于不好的参数设定和控制算法,使系统硬件面临烧毁的危险。

(2) 积分饱和现象会使控制品质变差。所谓积分饱和是指:对于具有积分运算的控制器,只要系统控制信号与被控量间存在偏差,PID 控制器的输出就会不停地变化来矫正偏差,但是当系统出现某种问题导致偏差一时无法消除时,控制器还试图矫正这个偏差,这样经过一段时间后就会造成控制器进入深度饱和状态。进入积分饱和的控制器只有等被控量偏差反向后才能逐渐消除,重新恢复控制作用。因此,在涉及控制器时,要考虑到抗积分饱和的问题。

(3) 微分近似。由于纯微分作用在实际中是不可能实现的,设计控制器时就要考虑如何对微分进行准确地近似。

(4) 在设计控制器时,经常会遇到无法确定需要使用哪种结构的问题,如 P、PI、PD、PID;需要确定使用一维还是二维的算法;需要确定是否需要输出饱和设置;需要考虑抗积分饱和特

性;特别是系统含有两个以上控制器时,还要考虑到算法切换时的瞬态稳定性。

(5) 如果要把 PID 控制器算法生成代码应用到实际系统中区去,还需要对算法和参数做离散化,定点化处理。

使用 PID 模块,可以很好地解决上述问题。下面通过为上文提到的直流电动机模型添加 PID 控制模块介绍其使用方法。

打开电动机模型,装载变量到工作空间,添加 math operations 库中的 Sum 模块到模型中,如图 2.2.60 所示。

双击模块打开参数设置对话框,将其符号设置为|+-,如图 2.2.61 所示。

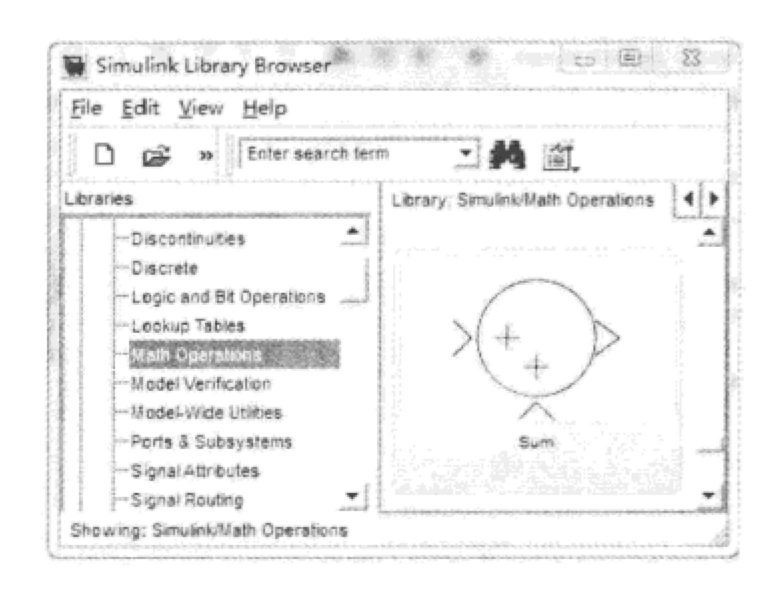


图 2.2.60 Sum 模块

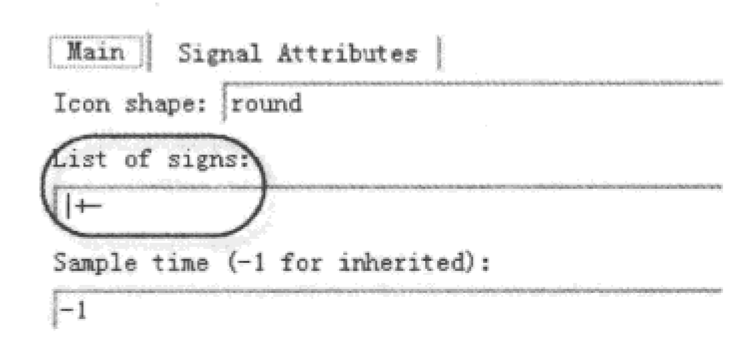


图 2.2.61 Sum 设置

添加 continuous 库中的 PID 模块到模型中,该模块位于图 2.2.62 所示的库中。

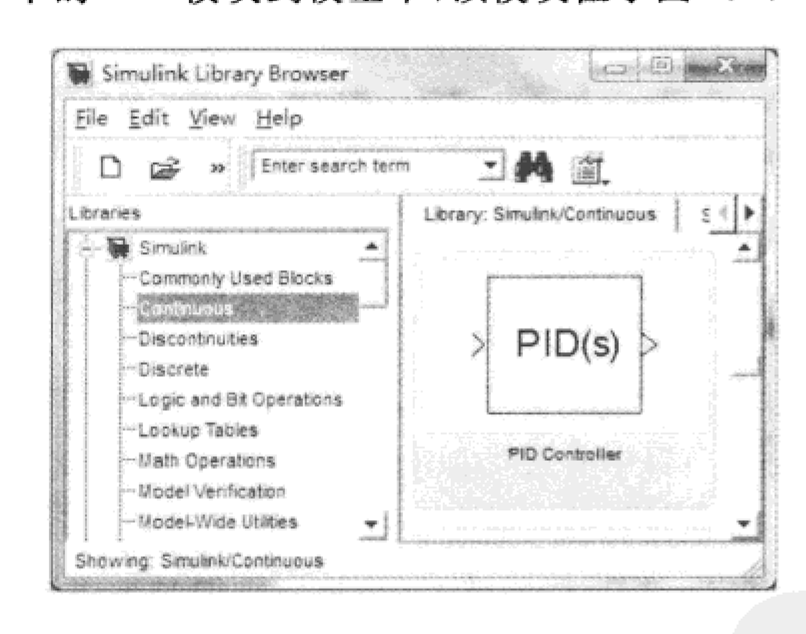


图 2.2.62 PID 模块

连接模型如图 2.2.63 所示。

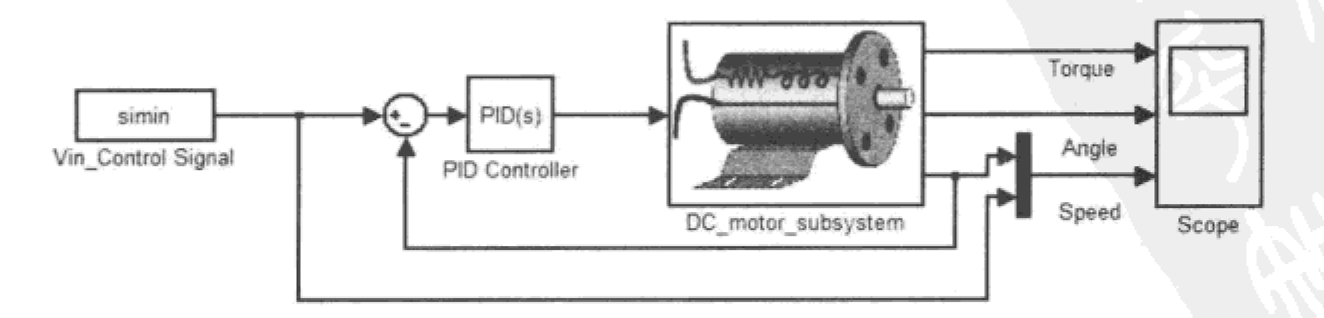
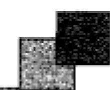


图 2.2.63 添加 PID、Sum 模块



1. 用 PID Tuner 设计 PID 控制器

双击打开 PID 模块的设置界面,如图 2.2.64 所示。

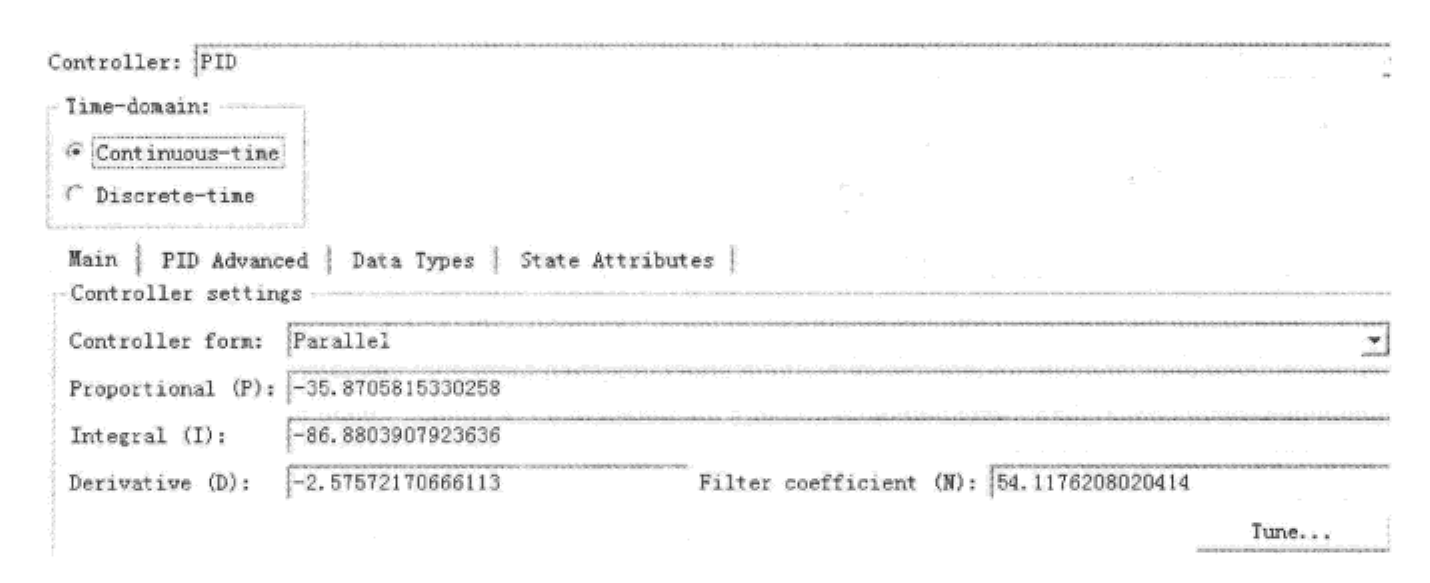


图 2.2.64 PID 模块设置页面

(1) Controller 通过 Controller 下拉菜单中的 P、I、PI、PD、PID 选项轻松地切换控制器的结构模式。

(2) Time-domain 选项可以快速地在连续域和离散域间切换。

(3) Controller Setting 选项可以为控制器选择工作状态"并行"或"理想型"具体区别可参考 help 文件。

(4) PID Advanced 选项卡中可以对控制器做一些高级设置,比如输出饱和限制,抗积分饱和特性,追踪模式等,如图 2.2.65 所示。

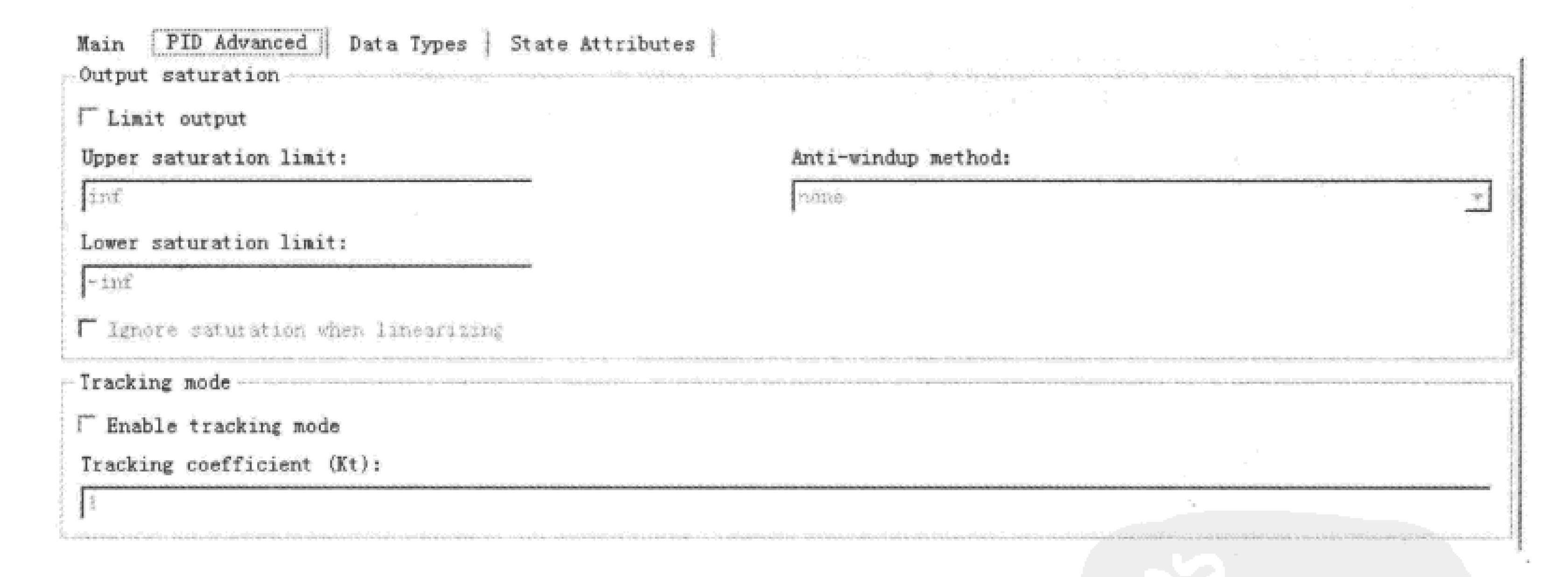


图 2.2.65 PID Advanced 选项卡

(5) Limit output:勾选复选框使能输出饱和限制。

(6) Anti - windup method:选择抗积分饱和模式。

(7) Enable tracking mode:勾选复选框使能追踪模式,使算法切换时更加平滑。

下面是最重要的一步:设置 P、I、D 的系数。传统的手动调节法不仅费时费力,还有可能对系统造成破坏,并且用户永远不知道自己设计出的参数是否为最优。

而基于规则的调节法无法应用于开环的不稳定系统,也不能用于高阶系统和有延迟的系统,且对用户的控制理论背景要求很高,不易掌握。

Simulink 提供了一个全新的 GUI 调节算法可以方便地完成这些复杂的工作，自动调节控制器参数以达到所期望的性能指标，通过简单的滚动条操作完成微调的功能。

单击参数设置页面上 Controller settings 中的 Tune 按钮打开 GUI 界面(单击 show parameters 可以完整地显示参数)，如图 2.2.66 所示。

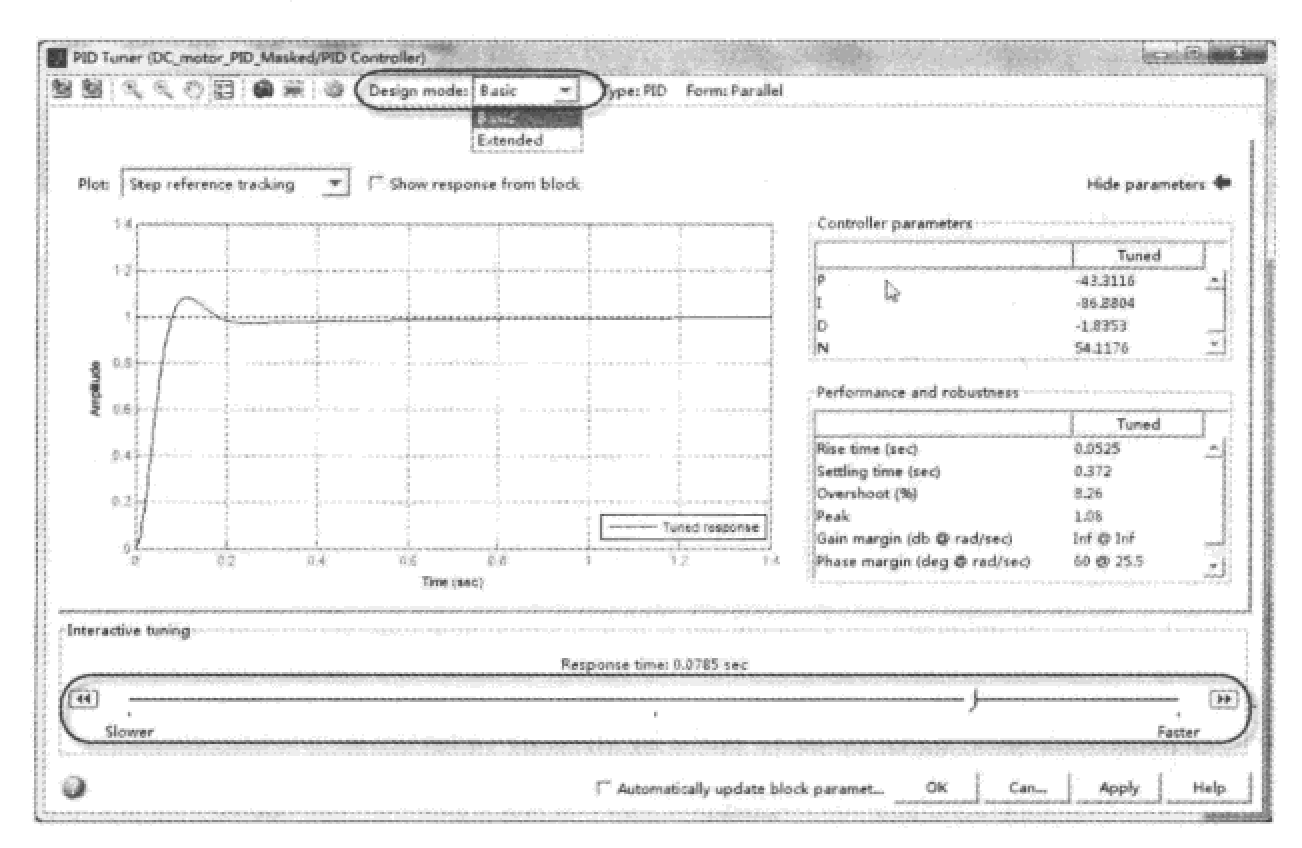


图 2.2.66 PID 自动调节界面

PID Tuner 会自动在系统默认的工作点处对模型进行线性化处理，设计出控制器的参数。用户可以直观地通过 GUI 界面看到系统的响应。

界面的下方有一个滚动条工具，通过拖动滚动条可以调节系统的响应时间，这里将响应时间调节到 0.0785s。通过 GUI 界面可以看到系统响应速度明显提高，并出现了一个小的过冲。

如果在界面上方的 Design mode 下拉菜单中选择 extended 命令，会出现额外两个滚动条“带宽”和“相位裕量”，通过拖动它们可以改变系统响应的快速性和平稳性，如图 2.2.67 所示。

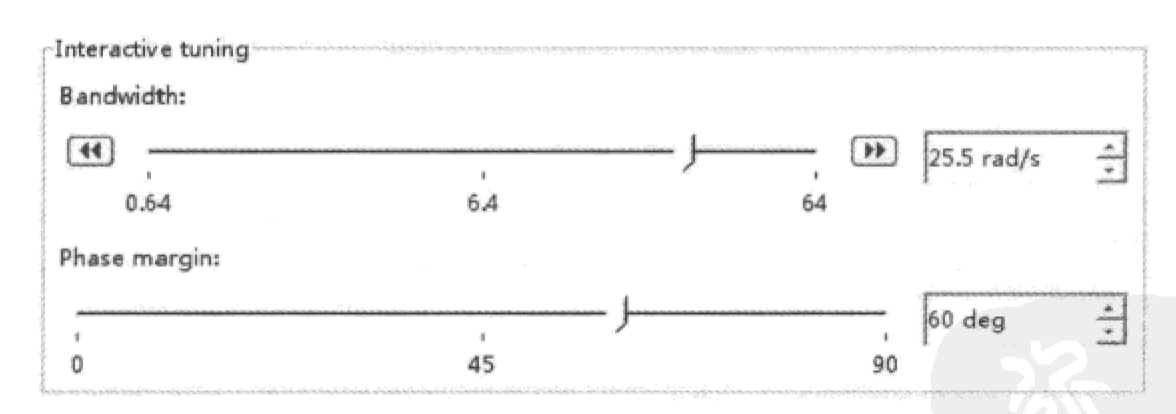


图 2.2.67 调节滑块

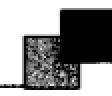
在显示阶跃响应曲线的区域内右击，根据需要选择 characteristics 中的一项或几项，会在响应曲线上添加相应的蓝点来表示这些特征点：

(1) Peak Response：峰值。

(2) Setting Time：稳定时间。

(3) Rise Time：响应时间。

(4) Steady State：稳定状态。



单击这些蓝点会显示其详细信息，如图 2.2.68 所示。

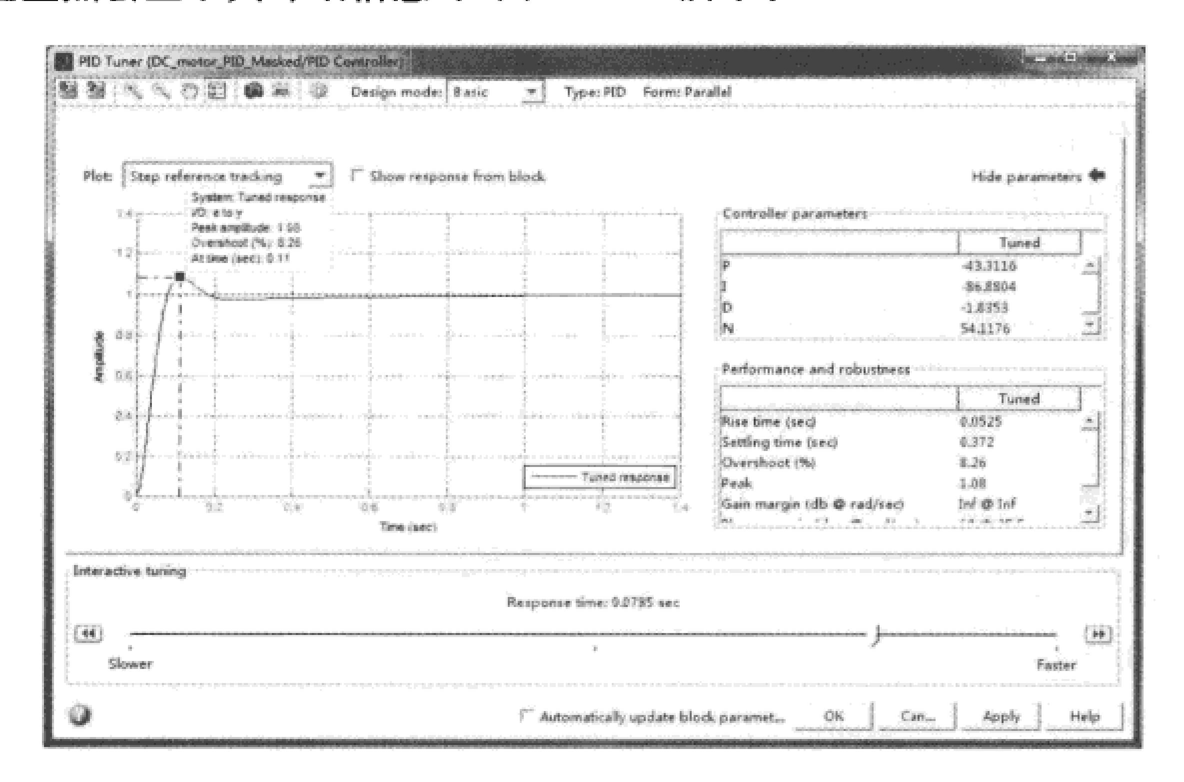


图 2.2.68　定位特征点

当用户得到满意的响应曲线后，单击 Apply 按钮，PID Tuner 自动设计的参数就已经写入到了参数设置框中，如图 2.2.69 所示。

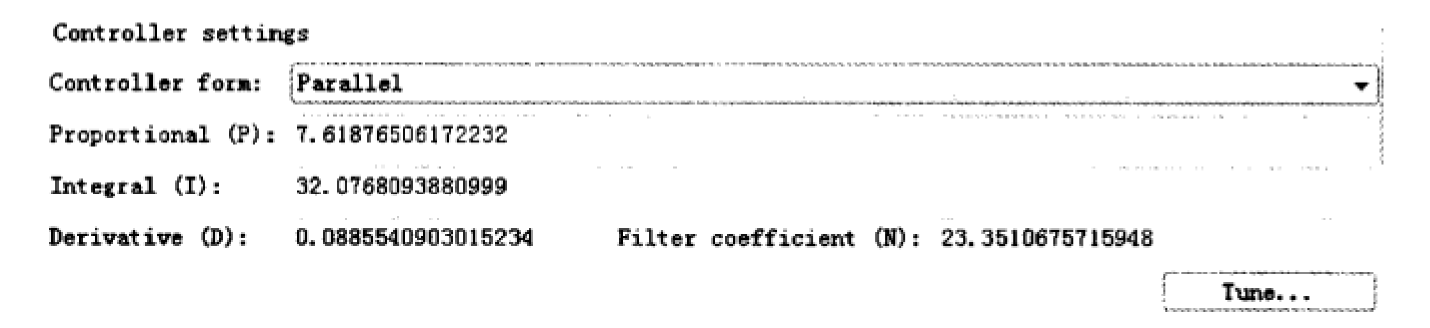


图 2.2.69　PID 参数

运行加入 PID 控制器后的直流电动机模型。可以看到，响应速度得到了很大提高，被控量转速 $\omega(t)$ 基本上已经和控制信号 V_{in} 基本吻合了，如图 2.2.70 所示。

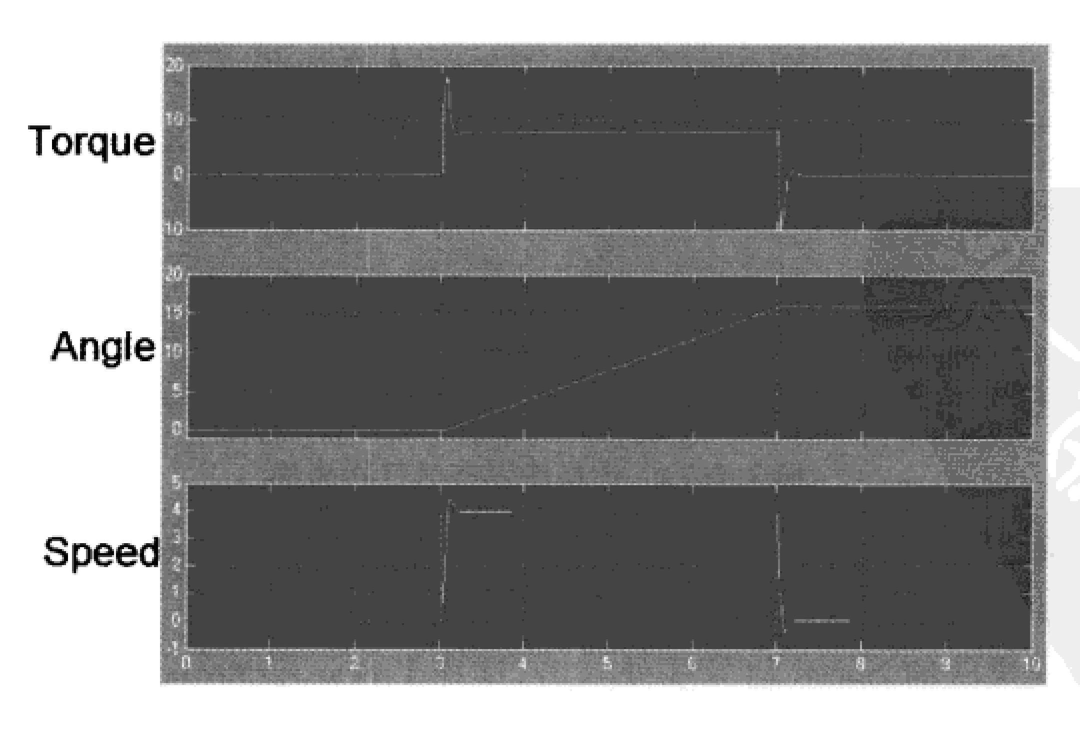


图 2.2.70　PID 调节后的仿真结果

2. 用波特图法设计 PID 控制器

Simulink 的 Control Design 工具提供了更多设计 PID 控制器的方法。这里以波特图设计法为例介绍如何用 Control Design 工具设计 PID 控制器。

打开添加过 PID 模块的电动机模型，选择 Tools→Control Design→Compensator Design 命令打开 Control and Eatimation Tools Manager 窗口。选择 Simulink Compensator Design Task 节点，单击 Select Block 按钮，确定将要调节的模块，如图 2.2.71 所示。

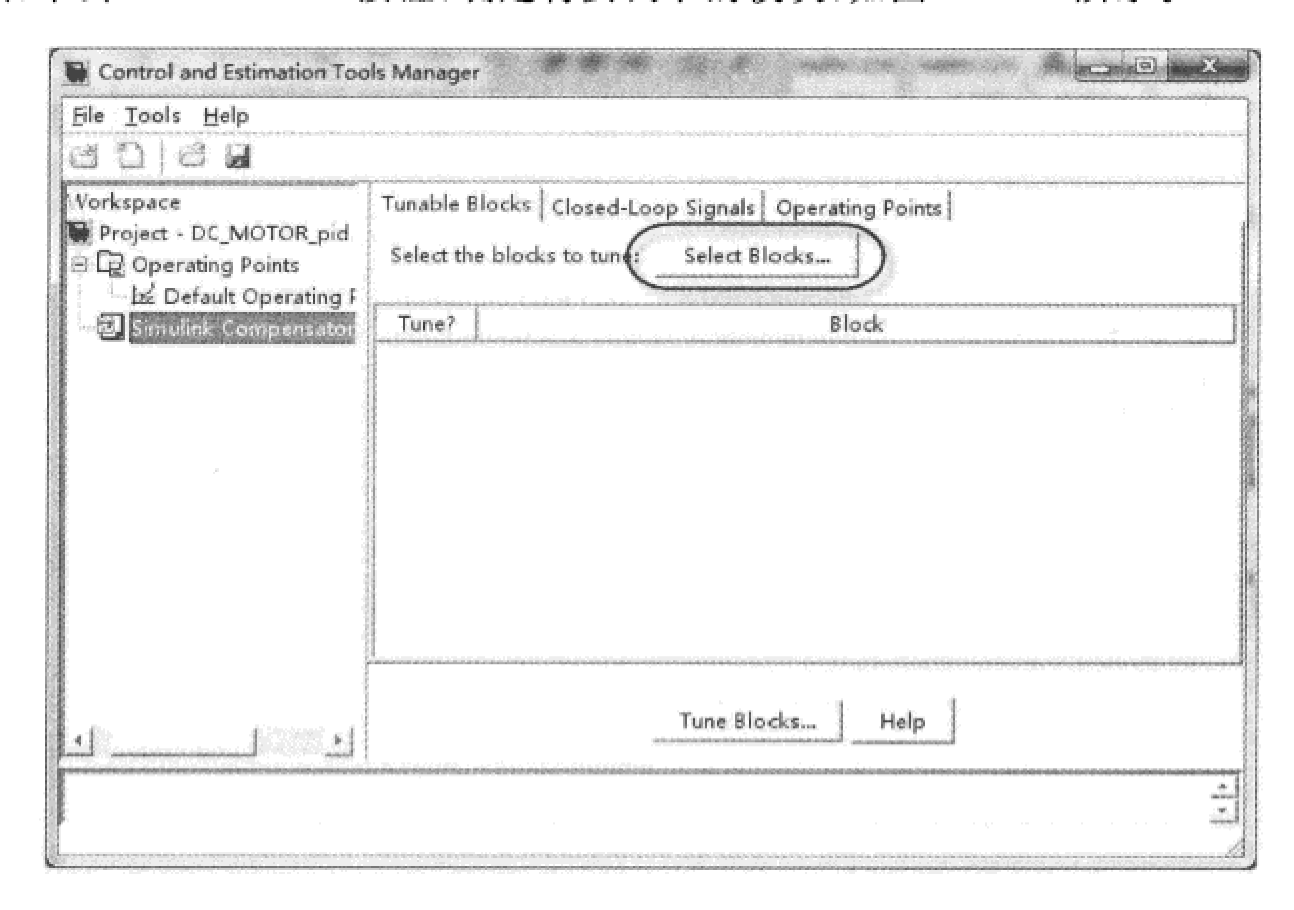


图 2.2.71 控制与估计管理工具界面

在弹出窗口中勾选 Tune? 复选框，单击“确定”按钮，如图 2.2.72 所示。

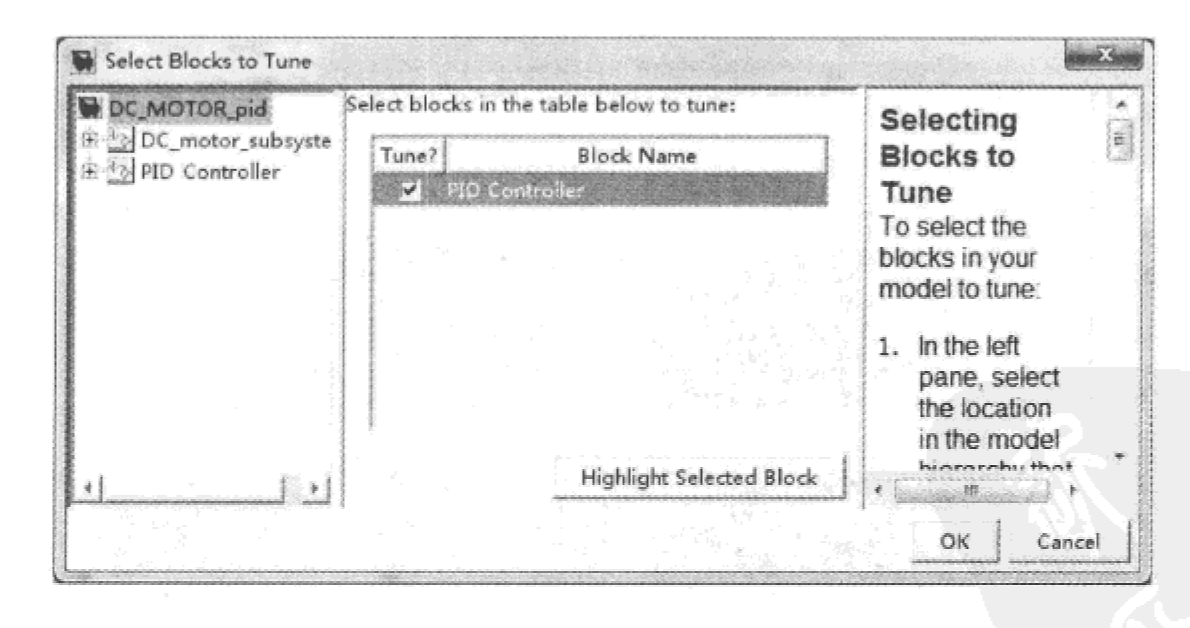
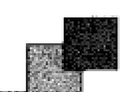


图 2.2.72 选择待调节的目标模块

设计 PID 控制器前，要先为系统指定需要调节的闭环系统，标识其输入/输出信号。在 Vin_Control Signal 模块后的信号线处右击，选择 Linearization points→Input Point 命令；在 DC_motor_subsystem 模块后的 Speed 信号线处右击，选择 Linearization points→Output Point 命令。这将在信号线处添加上 和 ，它们分别标识该闭环的输入和输出，如图 2.2.73 所示。



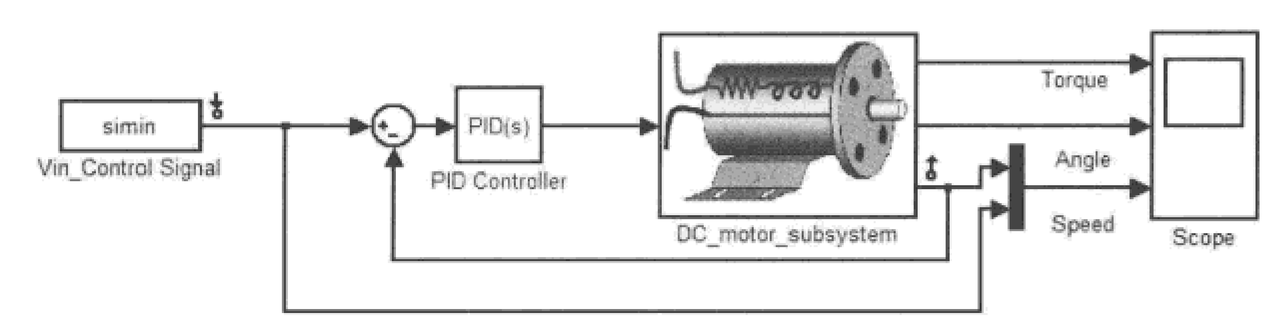


图 2.2.73　标记系统输入/输出

在 Control and Estimation Tools Manager 中单击 Tune Blocks 打开 Design Configuration Wizard 对话框，直接单击 Next 按钮，开始设置。

Wizard 的第一步是选择调节控制器所用到的设计图形，这里使用其默认配置并单击 Next；第二步选择分析响应曲线所用到的图形的类型，在 Analysis Plots 区域选择 Step 作为图形类型，在 Content in Plots 选项区域中勾选 1 复选框，用来画出从 Vin_Control Signal 到 DC_motor_subsystem 的闭环阶跃响应曲线，如图 2.2.74 所示。

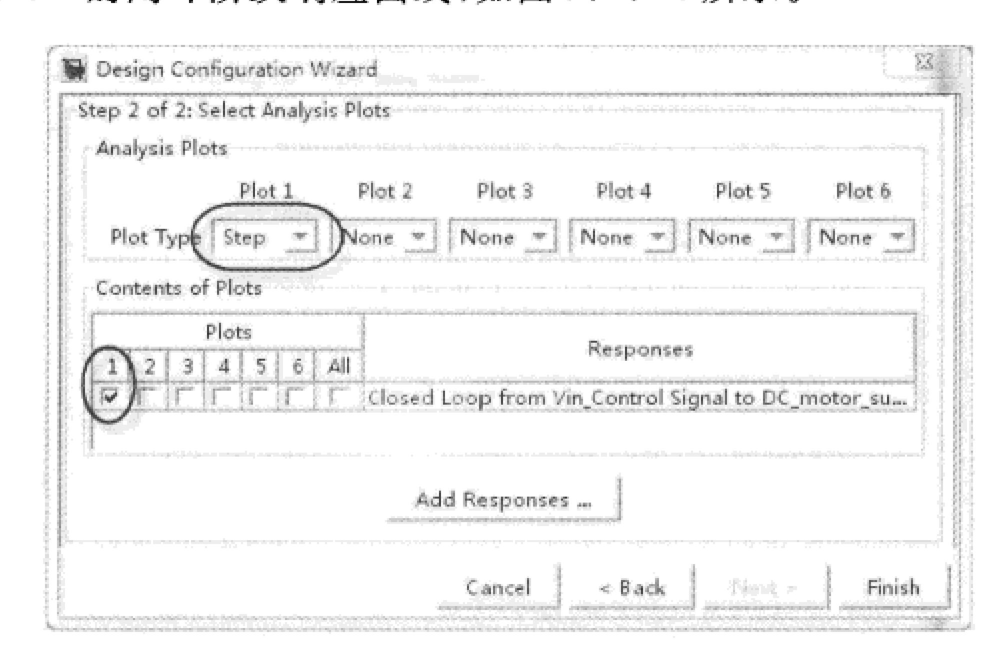


图 2.2.74　设置 step 类型

单击 Finish 按钮退出 Wizard，并画出阶跃响应曲线，如图 2.2.75 所示。

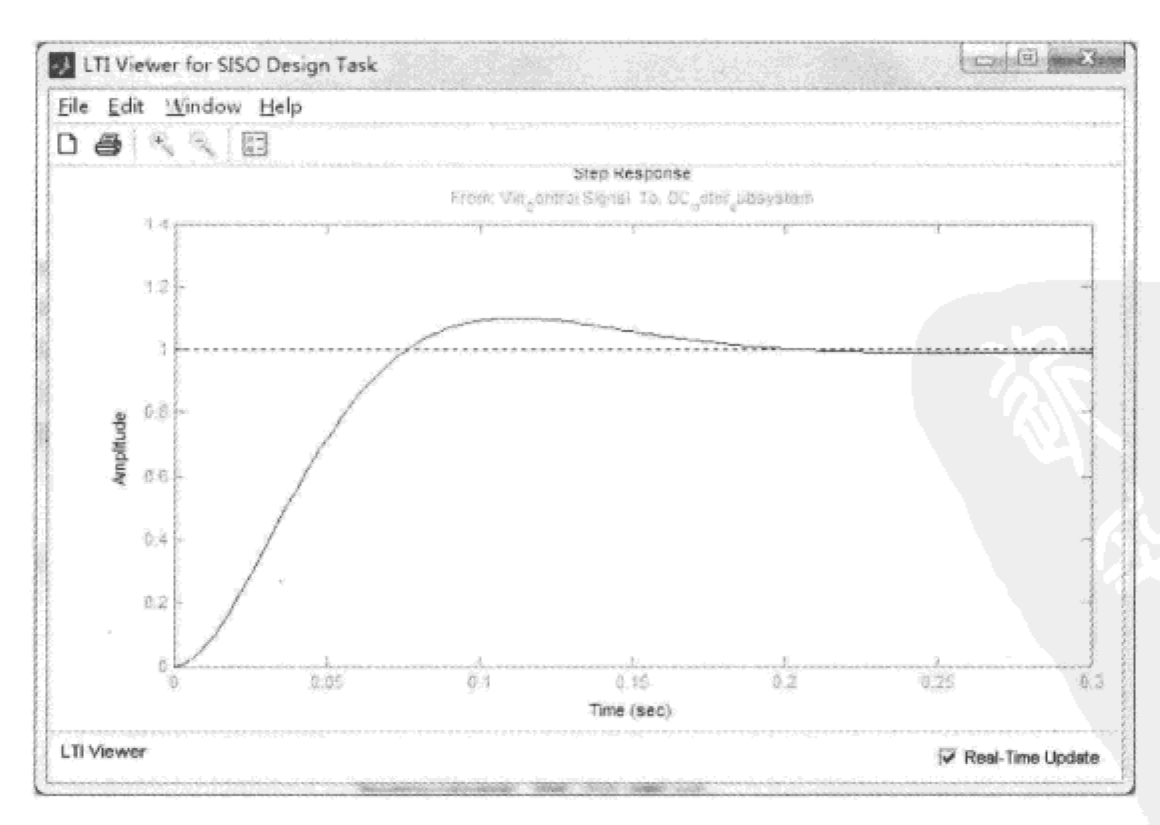


图 2.2.75　阶跃响应

这时还会弹出一个空白的 SISO Design for SISO Design Task 窗口,将在后面用到,不要关闭。

在 Control andEstimation Tools Manager 的 SISO Design Task 节点中打开 Graphical Tuning 选项卡,将 Plot Type 中 Plot1 的类型设置为 Open-Loop Bode。如图 2.2.76 所示。

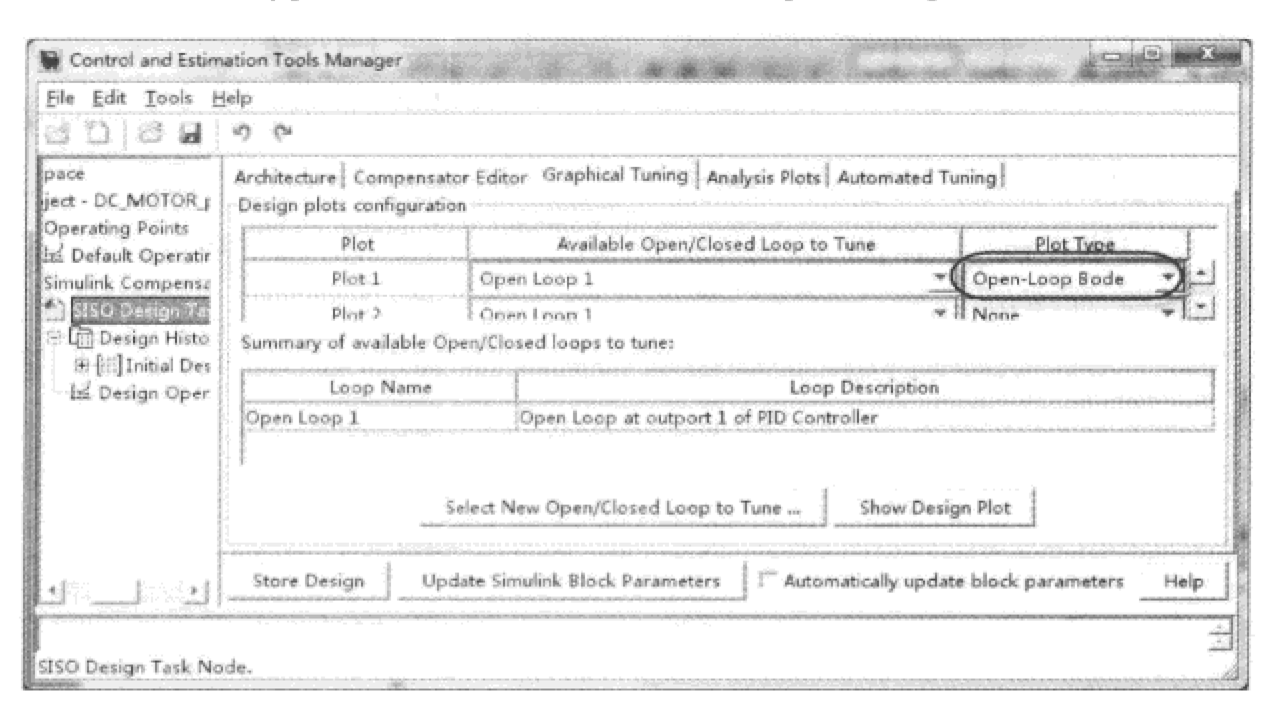


图 2.2.76　设置为开环波特图

完成上述操作后单击 Show Design Plot 按钮,在 SISO Design for SISO Design Task 窗口绘制出开环波特图,如图 2.2.77 所示。

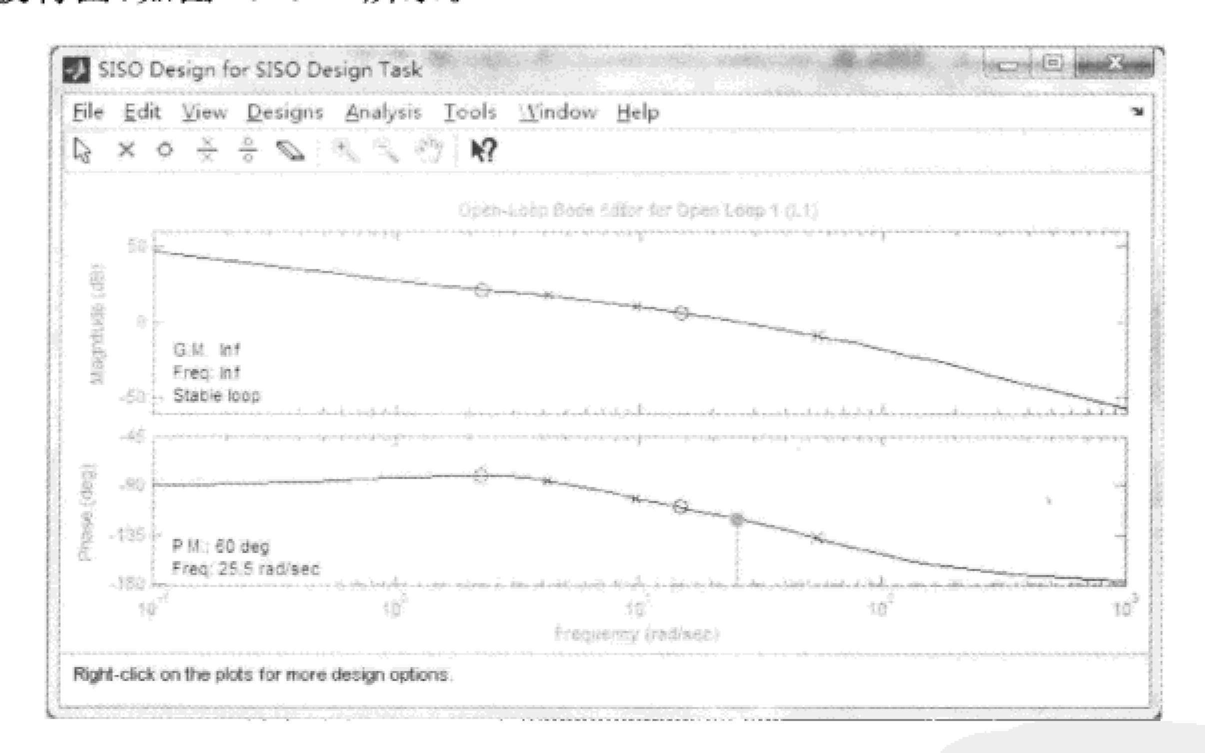
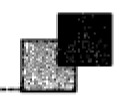


图 2.2.77　调节界面

图中的红色的圆圈和叉均可用鼠标拖动,曲线形状也会随之变化,同时,LTI Viewer for SISO Design Task 窗口中的阶跃响应曲线也会同步更新。例如,增大波特图中的增益,阶跃响应时间会缩短。

使用与 PID Tuner 设计 PID 控制器法相同的设置方式,同样可以显示调节得到的阶跃响应曲线的特征点数据,如图 2.2.78 所示。

当用户确定阶跃响应曲线符合要求时,可在 Control and Estimation Tools Manager 窗口的 SISO Design Task 节点中单击 Update Simulink Block Parameters 按钮。这样,设计的 PID 控制器参数即被写入到了 PID 模块中,设计过程完成。



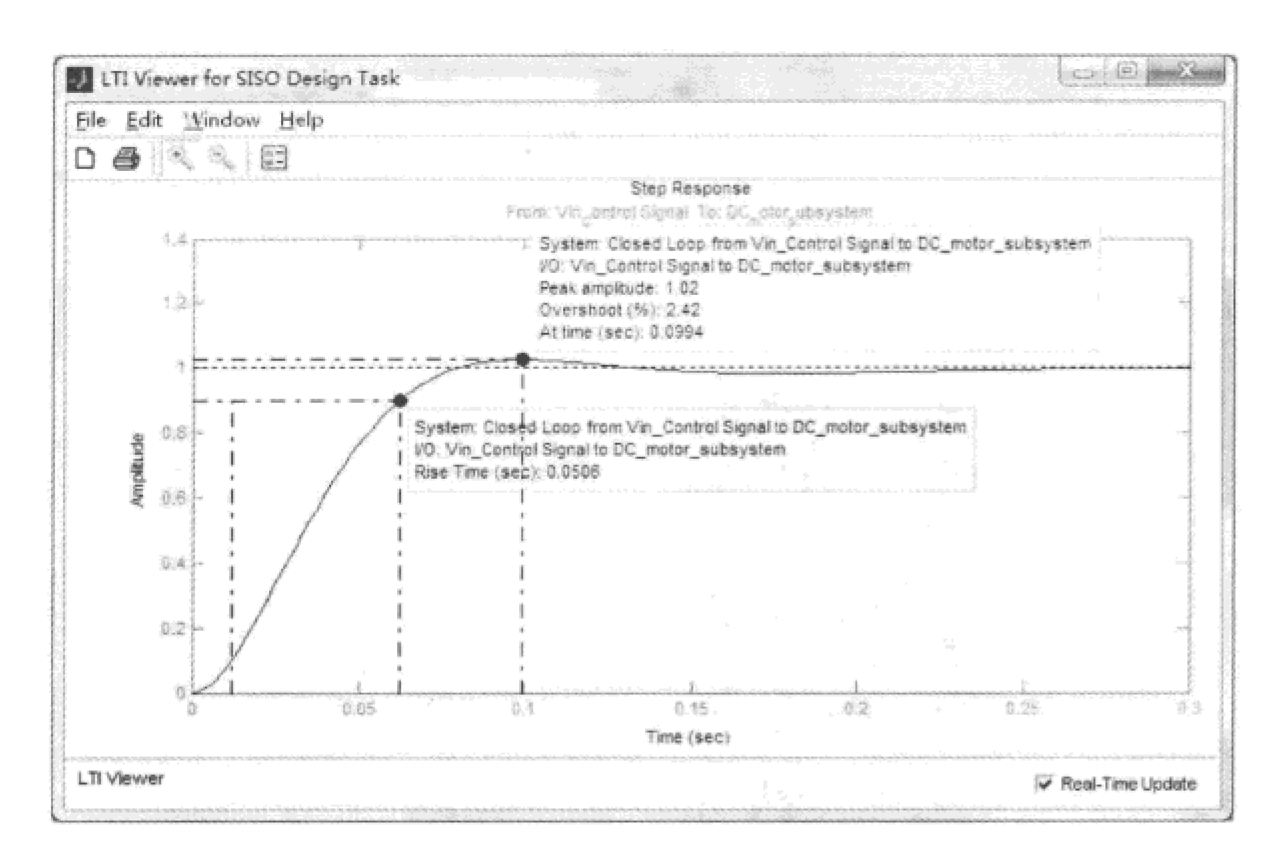


图 2.2.78 定位特征点

运行模型，观察电动机转速响应曲线，如图 2.2.79 所示。

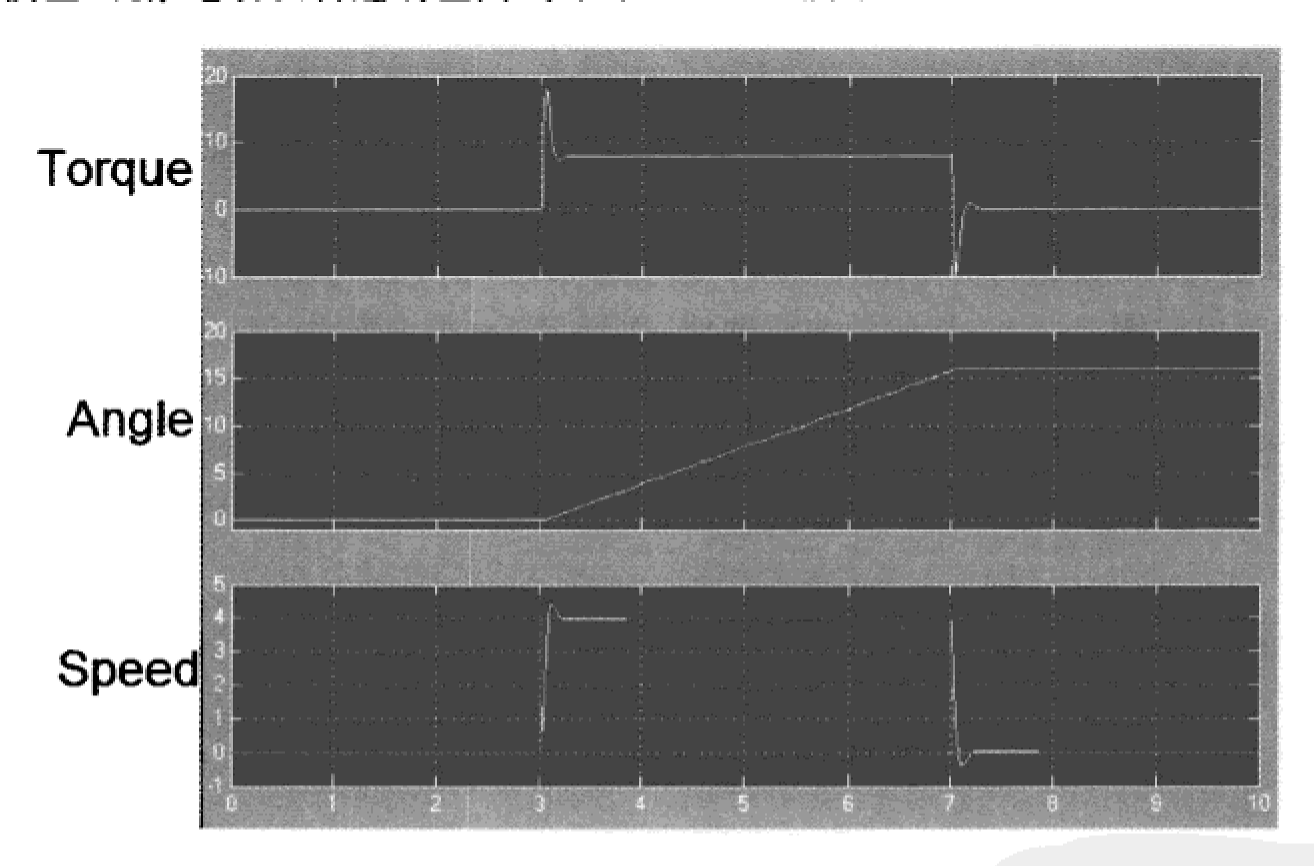


图 2.2.79 调节后的仿真结果

3. 离散域 PID 控制器

要把控制器算法应用到实际的控制器中，还需要将其转化到离散域，并且要考虑到实际处理器的位数和算法是否吻合。如果离散化与定点化做得不好，很可能导致一个在连续域性能不错的控制器转换到离散域、定点化后完全不能达到设计指标。

双击用上文方法设计好的 PID 模块，打开其设置界面，在 Time Domain 选项区域中点选 Discrete-Time 单选按钮，采样时间设为 0.1 s，单击运行模型，可以看到系统已经变得不稳定了，如图 2.2.80、图 2.2.81 所示。

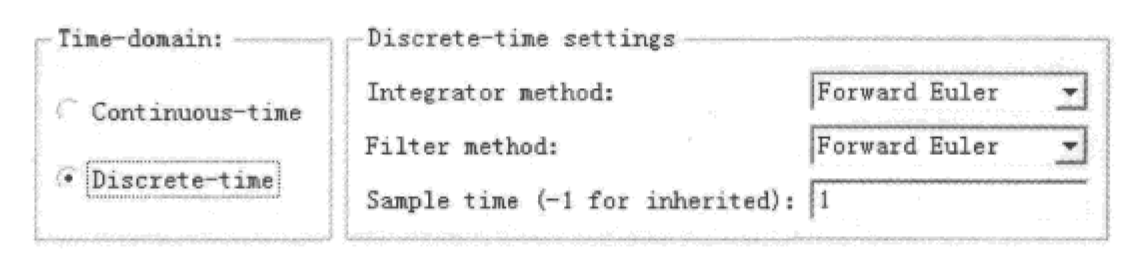


图 2.2.80 设置为离散 PID

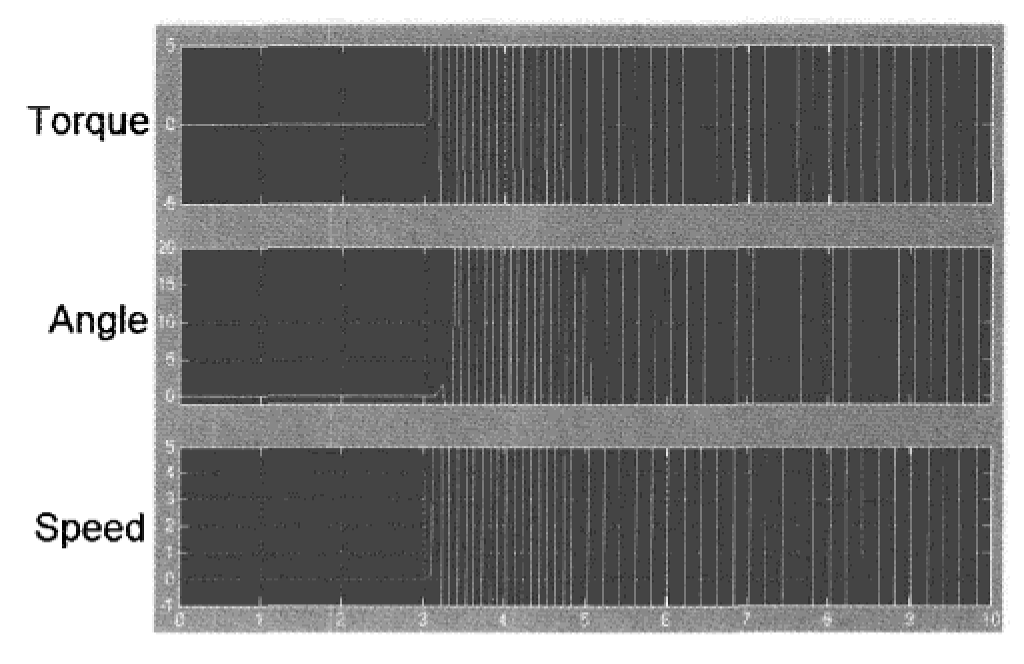


图 2.2.81 仿真结果

这是由于用户在用 PID Tuner 设计控制器时，为控制器设置的带宽为 25.5 rad/s（图 2.2.82），将其除以 2π 转化为 4.11 Hz，而采样带宽为 10 Hz，不到系统带宽的 3 倍。在实际工程应用中，采样带宽应该为系统带宽的 5～10 倍，可根据硬件系统的采样率来设置。这里假设为 0.01 s，单击 Tune 按钮打开 PID Tuner。按照本节"1. 用 PID Tuner 设计 PID 控制器"中介绍的方法调节出与图 2.2.66 相似的响应曲线，选择 Extended 模式，可以发现这时的系统带宽为 2.73 Hz，采样带宽是系统带宽的三十多倍，单击"确认"按钮，更新 PID 参数。这时如果手动调节到更大的系统带宽，则又会导致系统稳定性变差，如图 2.2.82 所示。

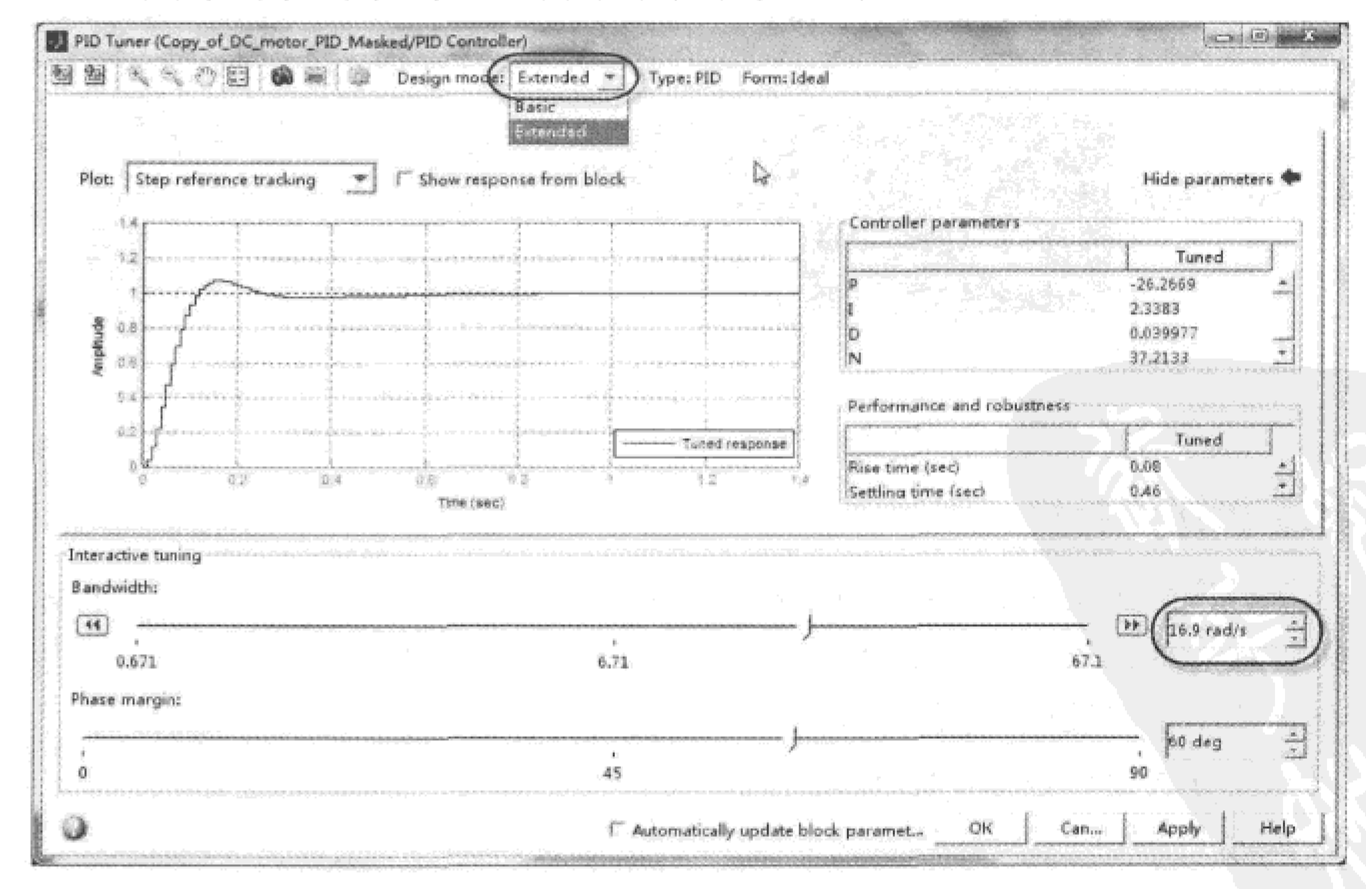
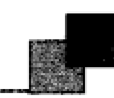


图 2.2.82 PID 调节界面



单击 OK 确认，更新 PID 参数，运行模型。这时系统再次变得稳定了，如图 2.2.83 所示。

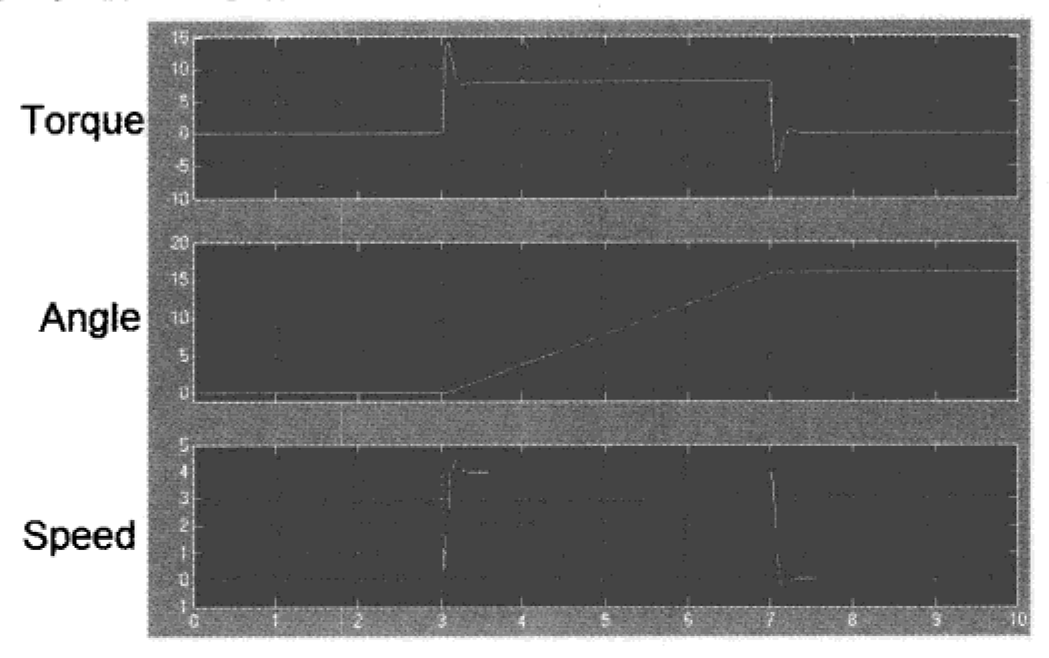


图 2.2.83　离散 PID 控制后的仿真结果

2.3　Simulink 模型调试

Simulink 为用户提供了功能强大的模型调试工具：Simulink Debugger。它是一个交互式的调试 Simulink 模型的工具。该工具可以设置断点，控制仿真的执行，显示模型的运行信息。通过使用调试器，用户可以一步一步地仿真模型，可以在任意步骤处单步运行，发现模型中可能存在的问题，或对其进行修改。

Simulink Debugger 有图形界面(GUI)和命令行界面两种运行方式：命令行界面模式可以实现调试器的所有功能，但需要用户事先了解调试命令；图形用户界面模式使用方便简洁，可以实现调试器的绝大部分功能，这里重点介绍图形用户界面模式。

2.3.1　图形界面调试

1. 启动调试器

图形界面调试器可以通过模型编辑窗口中的 Tools→Simulink Debugger 命令启动，也可以通过单击其工具栏上的图标“⊛”启动。调试器界面如图 2.3.1 所示。

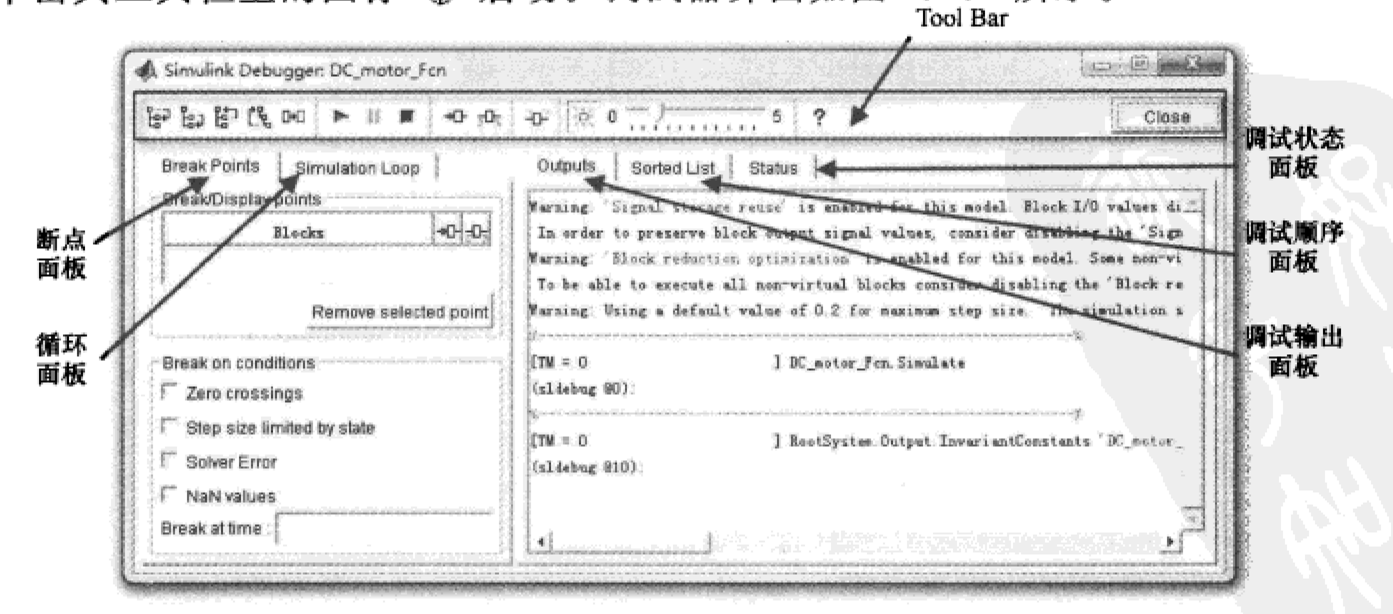


图 2.3.1　调试器界面

2. 图形界面调试器介绍

(1) 工具栏。工具栏位于调试器窗口的最上方,从左至右依次为以下指令:进入当前方法、跳过当前方法、跳出当前方法、在下一个方法开始时返回第一个方法、开始/继续运行仿真、暂停调试、终止调试、在选中的模块前设置断点、执行时显示选中模块的I/O信息、显示选中模块的当前I/O信息、开启/关闭动画、动画延时、帮助信息和关闭调试器,如图2.3.2所示。

这里提到的"方法"是指Simulink在仿真过程中逐时间步长对模型求解时所用到的运算方法。其实每一个模块都是由若干个这样的"方法"构成的,运行模型也就等价于对各个模块中的"方法"按一定的逻辑顺序逐时间步长运行。



图2.3.2 调试工具栏

(2) 断点界面。调试器提供了两种断点设置方法:一是普通断点,另一种是条件断点。当模型运行到用户设置的普通断点时,会无条件停止;而遇到条件断点时,则会判断是否达到指定的条件(过0,除0,限步长等),是则停止,否则继续运行。用户可以在断点界面中选择断点产生的条件。条件断点设置界面如图2.3.3所示。

① Zero crossing:模型在遇到过零点检测时产生断点。

② Step size limited state: 在状态手动条件约束时产生断点。

③ Solver error:求解错误时产生断点。

④ NaN value:仿真过程中遇到无限大,或溢出时产生断点。

⑤ Break at time:指定产生断点的具体时刻。

在断点产生条件选项框上面,调试器提供了一个断点信息显示框,它能显示当前系统中已设置断点的模块,并且能设置是否显示该断点的输入/输出。通过Remove Selected Point按钮可以取消相应的断点。

断点是非常有用的功能。特别是当用户知道模型中的某处可能存在问题时,通过设置适当的断点,可以迅速发现异常的模块和数据。断点设置的方法将在后面提到。

(3) 仿真循环界面。仿真循环界面显示"方法"的三列相关信息:Method、Breakpoints、ID。如图2.3.4所示。

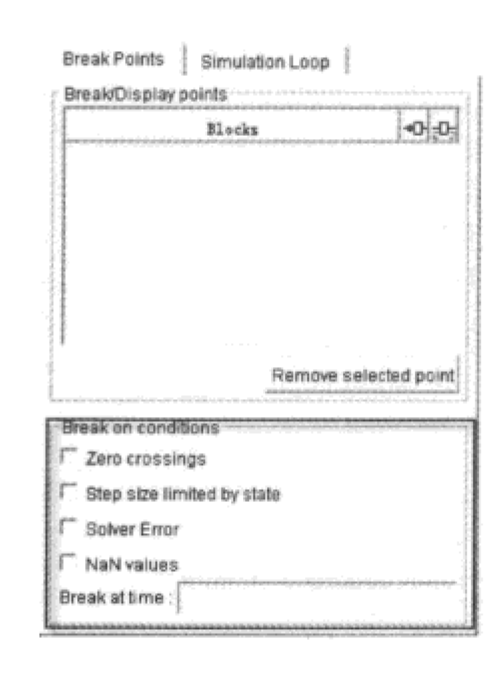


图2.3.3 断点设置界面

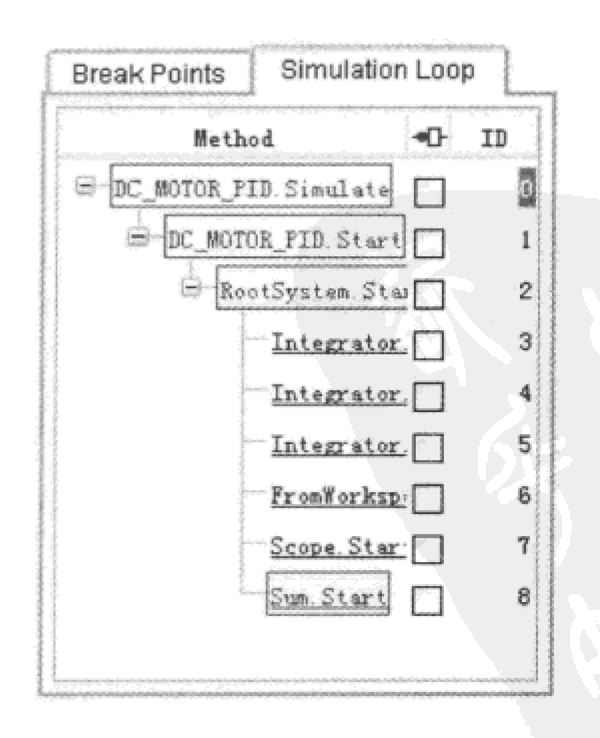
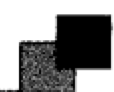


图2.3.4 仿真循环界面



① Method：方法名列以树形结构列出了从仿真开始时刻至此的所有“方法”。树形图的节点表示“方法”对其他“方法”的调用，展开一个节点即可观察上一级“方法”调用的次级“方法”。单击包含于模块中的“方法”(蓝色)会使模型中相应的模块高亮显示。

② Breakpoints：用户可以通过断点列的复选框设置断点。(在动画模式下此功能无效。)

③ ID：ID 列按顺序显示方法名列中所列出的所有“方法”的 ID 号码。

(4) 信息显示界面。信息显示界面共有 3 个选项卡：Outputs、Sorted Lists 和 Status。如图 2.3.5 所示。

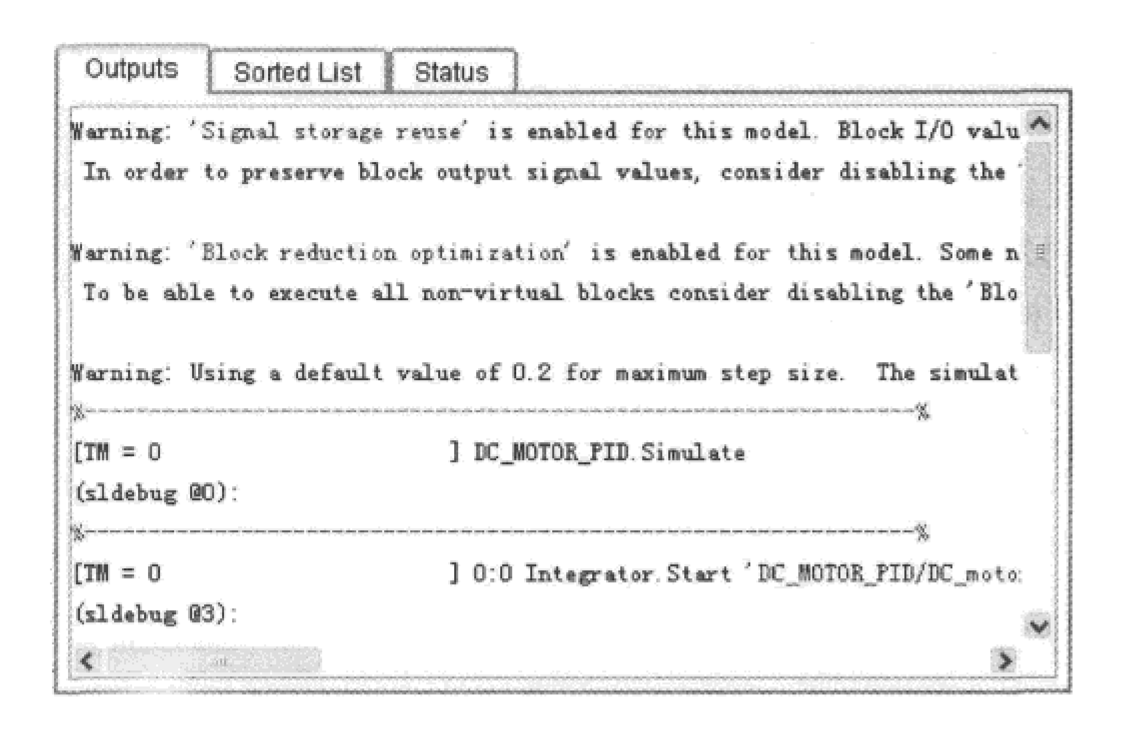


图 2.3.5　信息显示界面

① Outputs 选项卡：输出调试信息和结果，如调试命令、调试模块状态和该模块的输入/输出。

② Sorted Lists：按顺序显示调试过程中的非虚模块。

③ Status：显示各种调试设置的值和其他一些状态信息，如仿真时间、调试命令和调试断点等。

2.3.2　命令行调试

通过命令行模式调试可以调用调试器的所有功能，这对高级用户来说非常必要，也很方便。

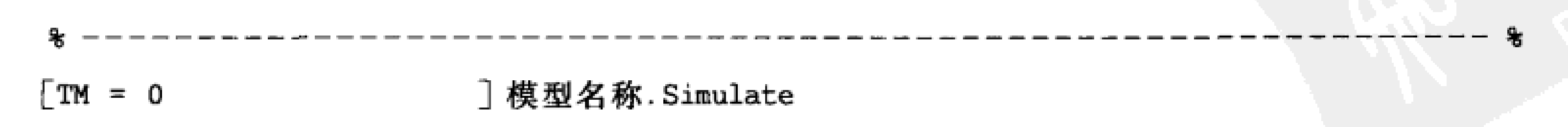
在 matlab 中的命令窗口中输入 sim 命令和 sldebug 命令都可以启动调试器：

```
>> sim('模型名称',[0,10],simset('debug','on'))
```

或者

```
>> sldebug '模型名称'
```

可以看到模块已经被打开，并在命令窗口中显示了如下信息：



```
%----------------------------------------------------------------------%
[TM = 0                    ] 模型名称.Simulate
```

此时调试器已经开始运行，用户可以继续输入其他调试指令、matlab 指令或者输入 stop 指令停止仿真调试。更多的调试命令可通过输入 help 命令获取。

2.3.3 运行调试器

在 GUI 模式下，单击“开始/继续”按钮▶开始进行调试。Simulink 可以从一个时间节点执行到下一个时间节点；从一个端点执行到下一个端点；或从一个模块执行到另一个模块，即单步调试。

1. 逐模块调试

这里仍然使用上文提到的直流电动机模型。打开模型，在 matlab 工作空间中定义参数，单击工具条中的▶按钮开始调试，模型上方会出现一个调试方法指示框@DC_MOTOR_PID. Simulate，提示用户模型已进入调试状态。

单击工具条中的 0+0 按钮执行单步运行，会看到有一个箭头由指示框指向模块，这个模块表示下一步即将执行的模块。以后每执行一次，指示框和箭头都会产生相应的变化，如图 2.3.6所示。

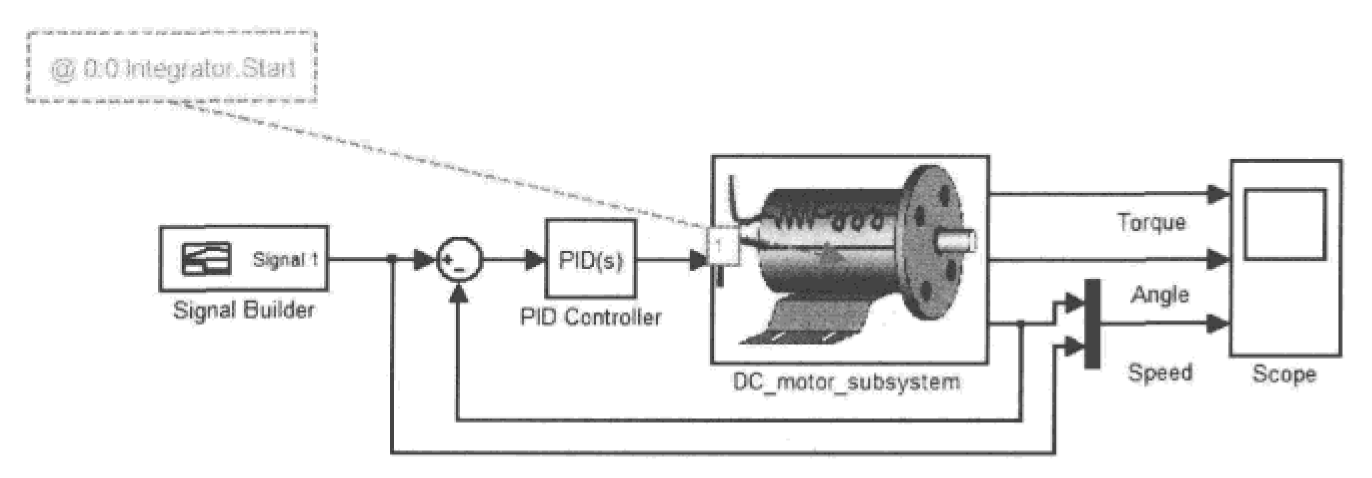


图 2.3.6 模型显示调试信息

在进行单步调试的同时，仿真循环界面中会以高亮显示下一步即将执行的模块。如图 2.3.7 所示。

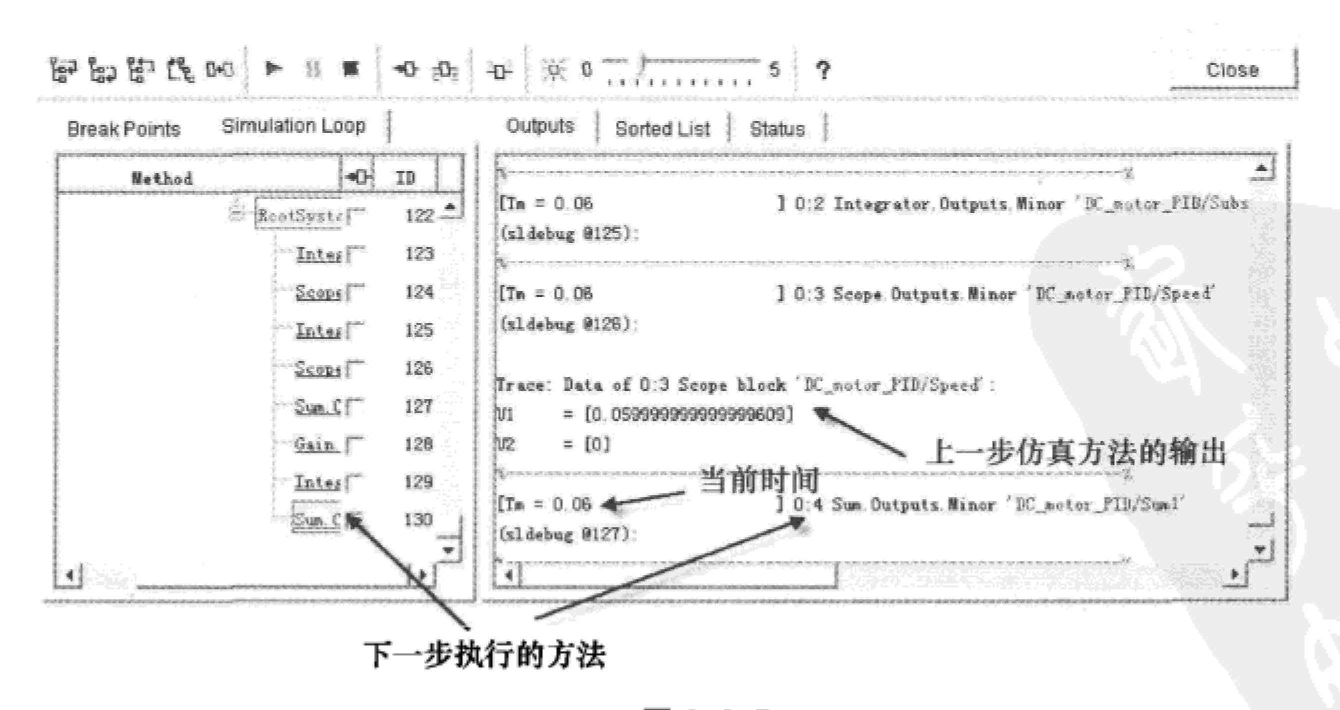
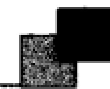


图 2.3.7



如果采用 Matlab 命令行方式调试，则在使用 sldebug 'DC_MOTOR_PID' 命令后输入 step 命令即可，命令窗口输出如下信息，TM=0 表示当前仿真时刻为 0；sldebug @0 表示即将被执行的模块为当前系统的第一个模块。

```
%--------------------------------------------------------------------%
[TM = 0                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> step
%--------------------------------------------------------------------%
[TM = 0                ] DC_MOTOR_PID.Start
(sldebug @1): >> step
%--------------------------------------------------------------------%
[TM = 0                ] RootSystem.Start 'DC_MOTOR_PID'
(sldebug @2): >> step
%--------------------------------------------------------------------%
[TM = 0                ] 0:0 Integrator.Start 'DC_MOTOR_PID/DC_motor_subsystem/Integrator'
(sldebug @3): >> step
Data of 0:0 Integrator block 'DC_MOTOR_PID/DC_motor_subsystem/Integrator':
CSTATE = [0]
%--------------------------------------------------------------------%
[TM = 0                ] 0:1 Integrator.Start 'DC_MOTOR_PID/DC_motor_subsystem/Integrator2'
(sldebug @4): >>
```

2. 逐时间步长单步调试

逐时间步长调试是按时间进程进行的，相对于逐模块调试，其调试效率较高，这对于复杂或模块数量较多的模型相当有效。

在 GUI 方式中，按下调试器上的绿色开始按钮 ►，启动调试，在未设置断点情况下，每按下按钮一次，即运行一次仿真步长，调试输出信息如图 2.3.8 所示。

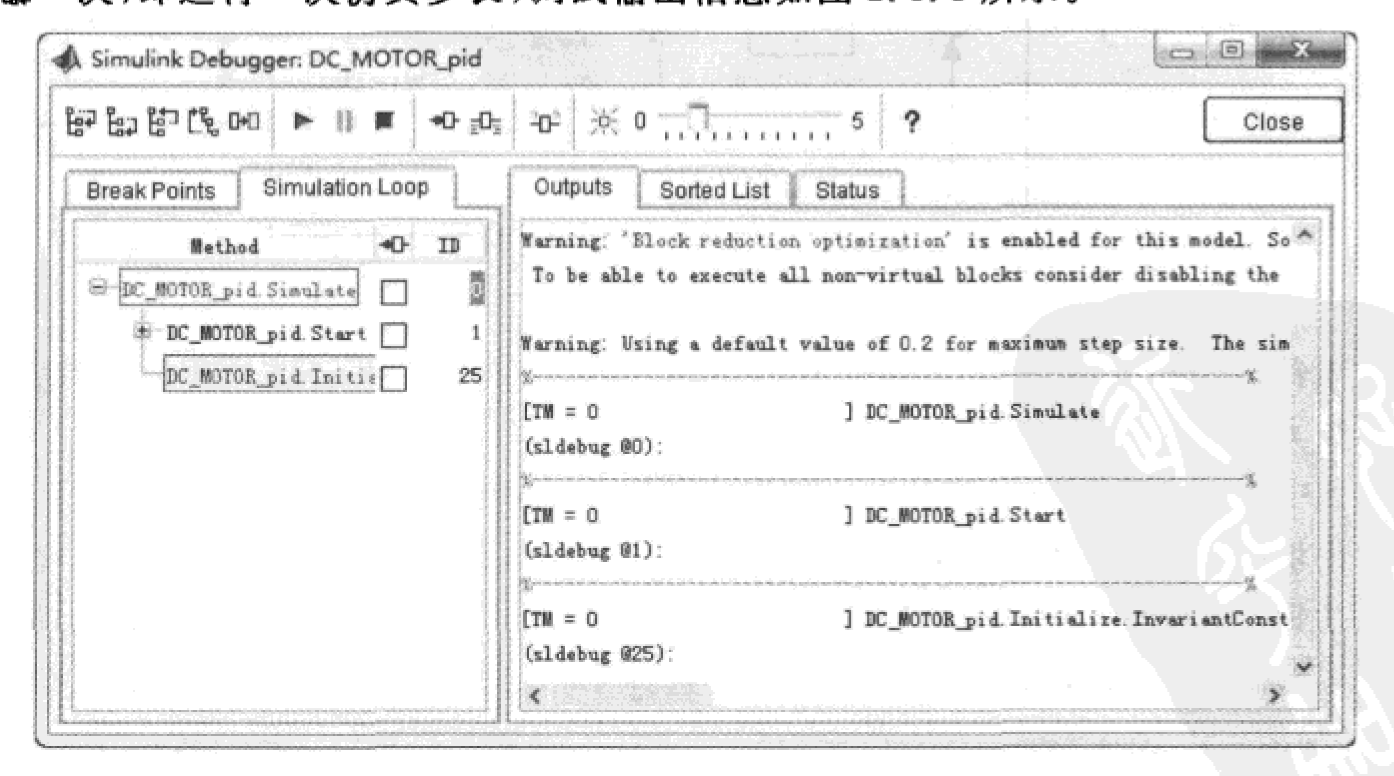


图 2.3.8　逐时间步长单步调试

对应命令行方式中的 next 指令，利用 next 命令可以执行当前时间步长中的所有模块（通常都大于一个），直接跳到下一个仿真时间步长。

```
% -------------------------------------------------------------- %
[TM = 0                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> next
% -------------------------------------------------------------- %
[TM = 0                ] DC_MOTOR_PID.Start
(sldebug @1): >> next
% -------------------------------------------------------------- %
[TM = 0                ] DC_MOTOR_PID.Initialize.InvariantConstants
(sldebug @25): >> next
% -------------------------------------------------------------- %
[TM = 0                ] DC_MOTOR_PID.Enable.InvariantConstants
(sldebug @27): >>
...........
```

3. 自动调试

在自动调试模式下，系统在当前仿真步长内，自动逐方法调试，每个方法间有一定的停顿，同时，用指针指向下一步运行的方法。

按下工具条中的 ▸ 按钮开始调试，单击按钮 ☼ 启动自动模式，通过滚动条可以调整停顿时间。运行结果如图 2.3.9 所示。

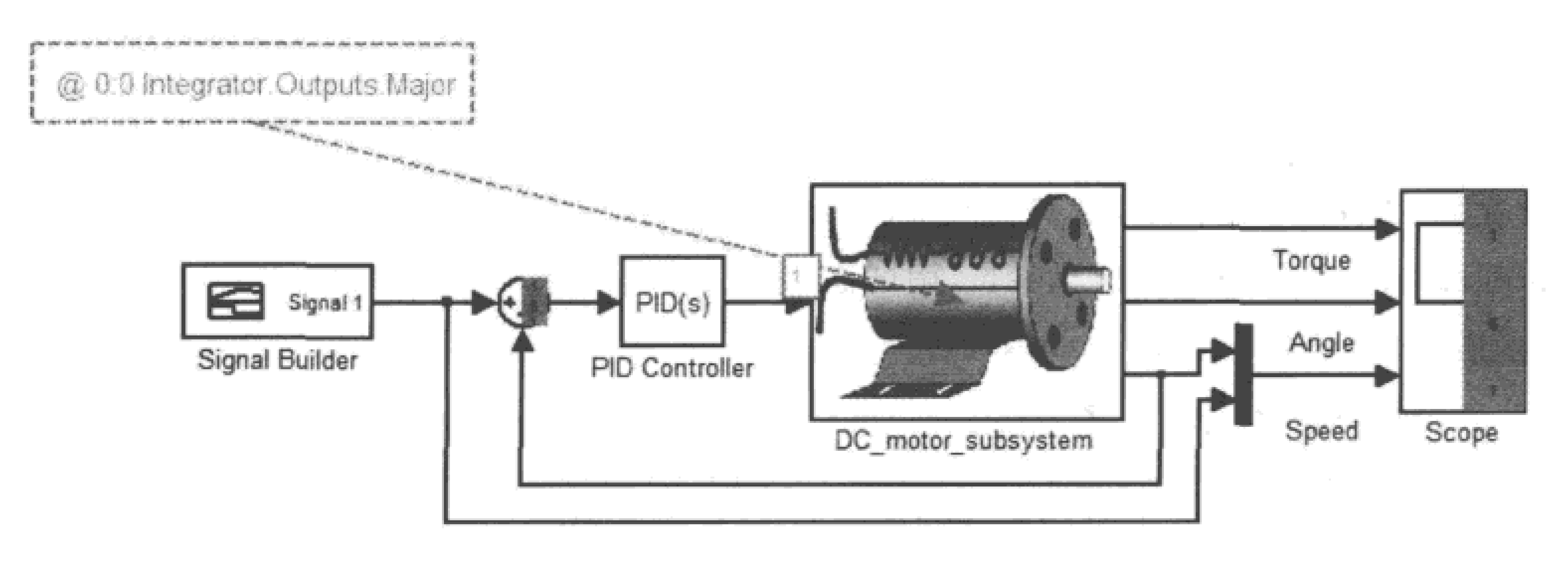
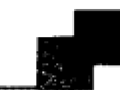


图 2.3.9 自动调试

调试指针指向的模块，会出现颜色不同、带有数字的小方块，各种颜色代表了不同的方法，数字表示该方法被调用的次数。

注意到 subsystem 和 PID Controller 模块并没有出现带颜色的方块，这是因为系统将其判定为虚模块，而模块指针只做用于非虚模块。打开 subsystem，可以看到其中的非虚模块已经被模块指针标注了调用次数，如图 2.3.10 所示。PID Controller 的情况类似，如图 2.3.11 所示。可通过右击模块，在弹出的右键菜单中选择 Look Under Mask 命令，打开其底层非虚模块。



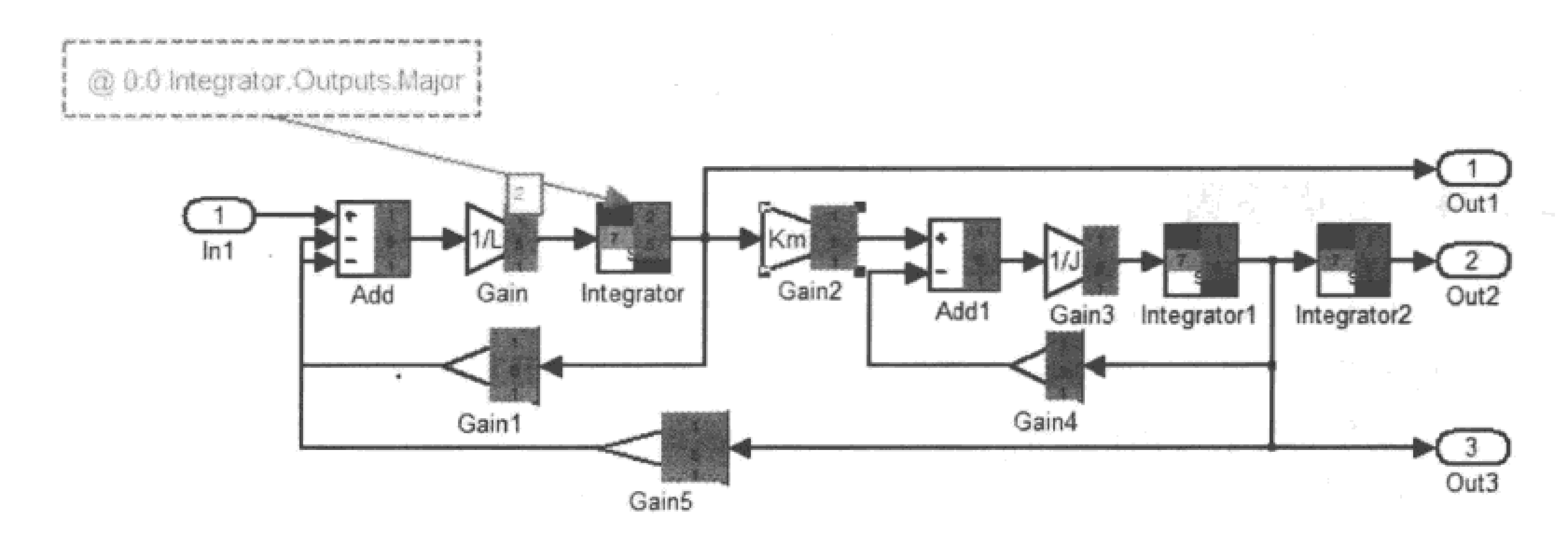


图 2.3.10　电动机子系统的调试信息

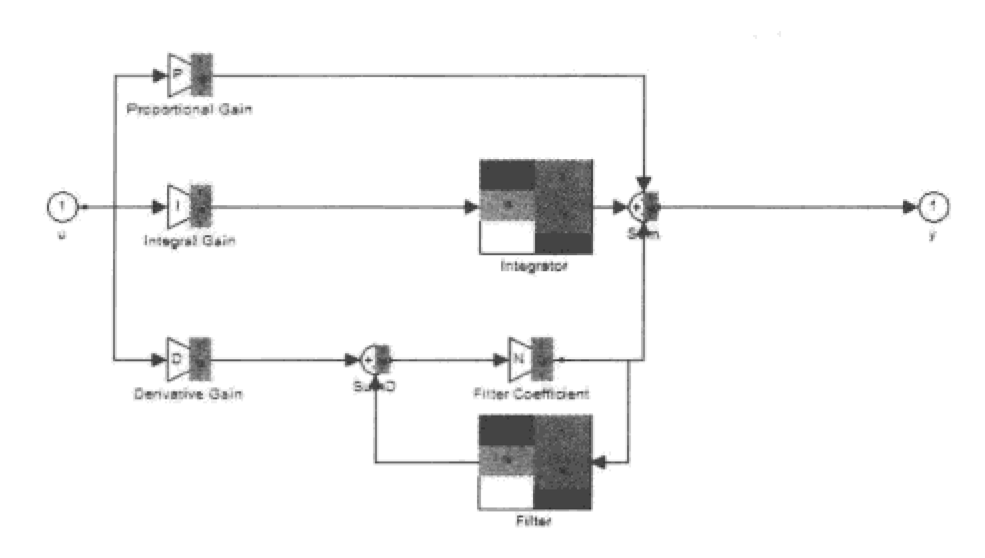


图 2.3.11　PID 系统的调试信息

4. 运行至断点

当确知模型中的某处可能存在问题时，断点是非常有用的，通过设置适当的断点可以快速确定异常的位置，或跳过异常执行后续仿真。

在积分器模块处设置断点后，单击工具条中的按钮 ▸ 输出调试信息如图 2.3.12 所示。

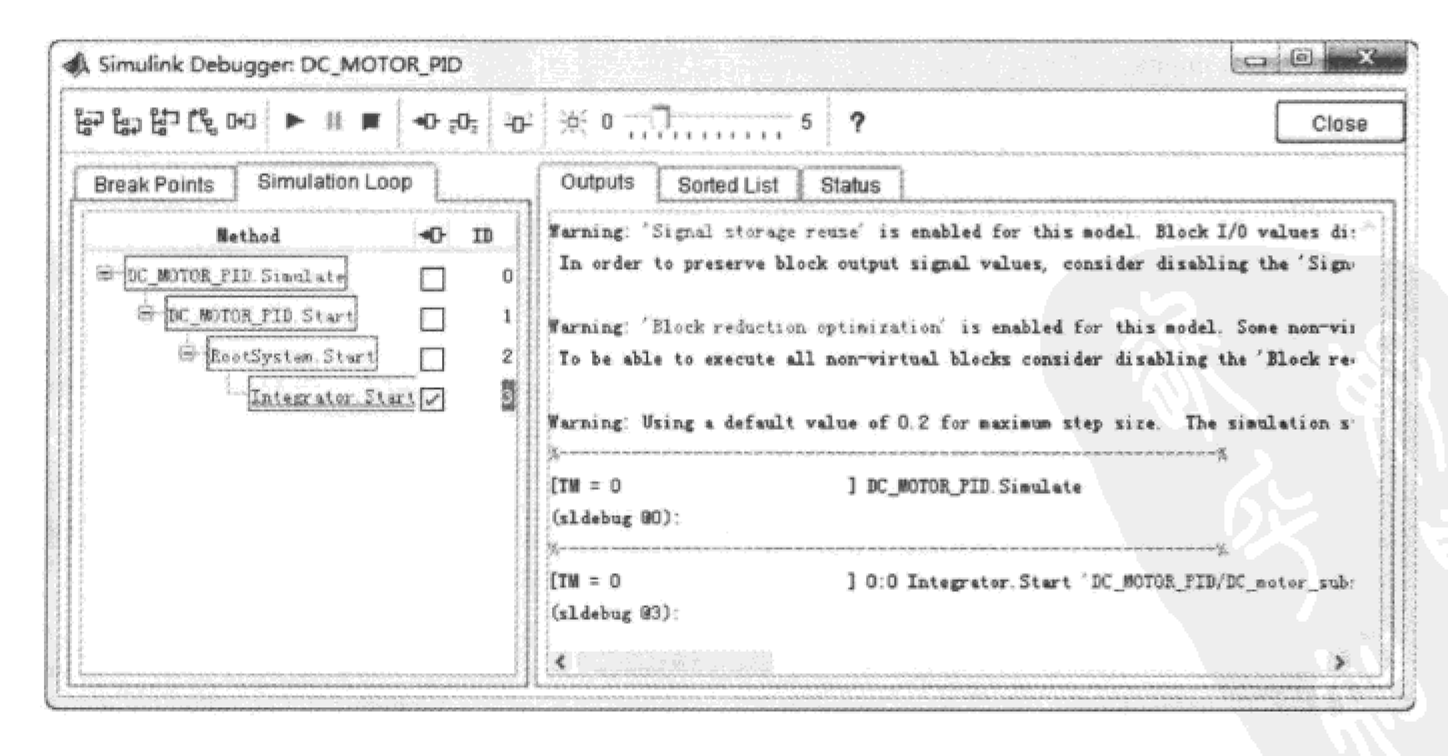


图 2.3.12　运行至断点

可见调试进行到第三个方法时遇到了断点，并暂停了调试。在命令行模式中，可以用 continue 命令从当前断点运行到下一个断点或仿真的终点。断点设置方法将在后面提到。

2.3.4 断点设置

借助断点可以快速诊断系统，找到仿真过程中存在的错误。Simulink 调试器允许用户针对不同情况设置不同类型的断点：无条件断点与有条件断点。

(1) 无条件断点：无论任何条件，仿真到达到预设断点处即中断运行。

(2) 有条件断点：若满足断点条件，仿真到达预设断点处即中断运行。

1. 无条件断点

无条件断点可通过以下三种方法设置：

- GUI 调试器工具栏快捷按钮。
- GUI 调试器的 Simulation Loop 页面。
- 命令窗口。

(1) 通过 GUI 调试器工具栏快捷按钮设置断点。在模型窗口选中需要设置断点模块(例如选择 Integrator 2)，然后单击图形调试器工具栏中的 Breakpoint 按钮设置断点。断点页面即显示当前断点，如图 2.3.13 所示。

用户可以在模块中取消勾选断点复选框，若要删除模块列表中的断点，可以单击 Remove selected point 按钮。

(2) 通过 GUI 调试器的 Simulation Loop 选项卡设置断点。此设置方法需要先运行调试器，当调试器仿真若干个模块后，Simulation Loop 选项卡中经历的模块才能显示出来，对于复杂系统，需要执行得更多步才能显示出期望的模块，如图 2.3.14 所示。

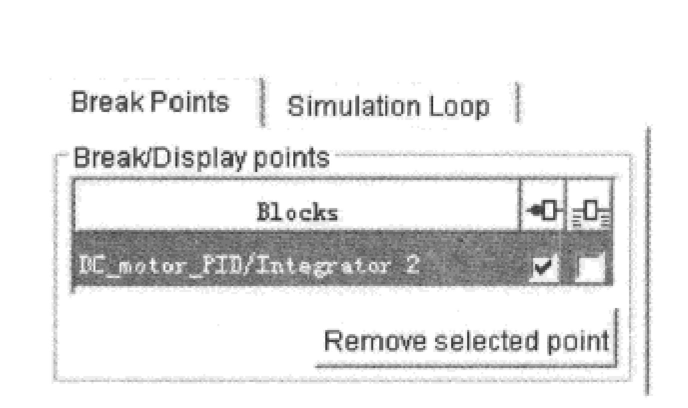


图 2.3.13 断点设置

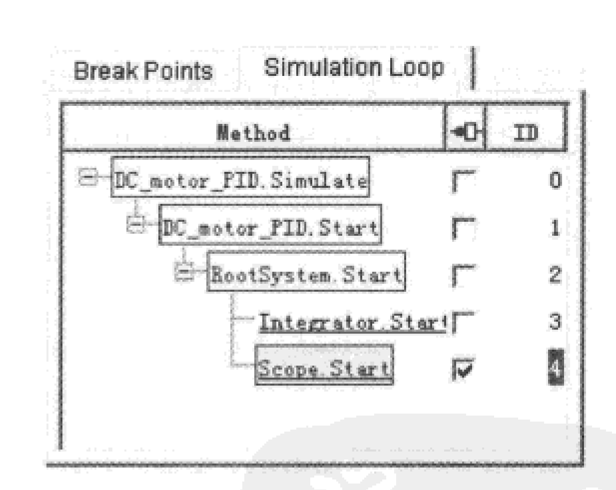
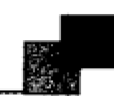


图 2.3.14 运行至断点

(3) 通过命令窗口设置断点。命令窗口设置断点的方法只适用于命令行调试模式下，在该模式下，命令 break 和 bafter 分别用来在模块的前端和后端设置断点，clear 命令则用来清除断点，命令行调试窗口的输出信息如下：

```
%----------------------------------------------------------------------%
[TM = 0                ] DC_MOTOR_PID.Simulate
(sldebug @0): >> step
```



```
% ---------------------------------------------------------------- %
[TM = 0                  ] DC_MOTOR_PID.Start
(sldebug @1): >> break
Installed break point:0 before m:1

(sldebug @1): >> bafter
Installed break point:1 after m:1

(sldebug @1): >> clear
Break point '0' has been removed.
Break point '1' has been removed.

(sldebug @1): >>
```

2. 条件断点

无条件断点的特点是：每当系统运行到无条件断点处时都会暂停。而条件断点处是否发生中断还不确定，取决于具体的运行情况是否符合条件，如图 2.3.15 所示。

Break on conditions
Zero crossings
Step size limited by state
Solver Error
NaN values
Break at time :

图 2.3.15 条件断点界面

在断点设置界面中的下部复选框中可以选择中断条件，每项的含义已在上文做过介绍：

其对应的命令行指令如下：

- zcbreak：仿真发生过零时设置断点。
- xbreak：仿真步长超过模型步长限制时设置断点。
- ebreak：求解出错处设置断点。
- nanbreak：数值溢出或无穷大时设置断点。

(1) Zcbreak。当需要对模型的过零情况进行中断时，勾选 Break on conditions 中的 Zero crossings 复选框或在命令窗口中执行 zcbreak 命令。暂停后显示该中断在模型中的 ID、类型、名称、时间，以及在过零时是上升还是下降。

在命令行调试模式下，使用 zcbreak 命令设置过零断点，continue 命令开始运行仿真，在下一断点或当前仿真步长末尾前停止运行，过零中断的输出信息如下：

```
% ---------------------------------------------------------------- %
[TM = 0                  ] DC_MOTOR_pid.Simulate
(sldebug @0):
1 Zero crossing detected at the following location
0 0:3:0 FromWorkspace 'DC_MOTOR_pid/Signal Builder/FromWs'
ZeroCrossing Events detected. Interrupting model execution
......
```

(2) Xbreak。该命令主要是针对变步长求解器，勾选 Break on conditions 中的 Step size limited by state 复选框，或在命令窗口中使用 Xbreak 命令。当求解器所采用的步长超过模型的步长限制时，仿真就会自动中断，这种中断方式主要用于调试那些有可能需要过大仿真步长的模型。

```
.........
[TM = 0                    ] DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> xbreak
Break on failed integration step             : enabled

(sldebug @30): >>
```

(3) Ebreak。勾选 Break on conditions 中的 Solver Error 复选框，或在命令窗口中输入 ebreak 命令，系统将在检测到一个可恢复的错误时产生中断。如果用户未设置此条件断点，系统将会自动修复错误，但是不会发出提示信息。

```
........
% ---------------------------------------------------------------- %
[TM = 0                    ] DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> ebreak
Break on solver error                        : enabled

(sldebug @30): >>
```

(4) NaNbreak。勾选 Break on conditions 中的 NaN values 复选框，或在命令行调试模式输入 nanbreak 命令，系统将在求解器计算出一个无穷大数值或溢出时产生中断。设置这类中断，可以精确地找到系统中的计算错误。

```
..........
% ---------------------------------------------------------------- %
[TM = 0              ] DC_MOTOR_PID.Output.InvariantConstants
(sldebug @30): >> nanbreak
Break on non-finite (NaN,Inf) values         : enabled

(sldebug @30): >>
```

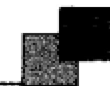
(5) Tbreak。在 Break on conditions 的 Break at time 文本框输入时间，或使用 tbreak 命令设置时间断点，系统会将在指定的时刻的下一个步长处中断并停留在模型的 Outputs. Major 方法。

2.3.5 显示模型和仿真信息

1. 显示模块的输入/输出信息

通过调试器工具栏上的按钮和按钮显示模块的输入/输出。用于命令行模式的相应指令由 probe、disp、trace。调试命令 trace gcb 与 probe gcb 分别对应于上述两个按钮，但 probe 和 disp 命令并没有与之相对应，下面解释其不同之处。

(1) 设置模块的输入/输出显示点。在 GUI 界面下，选中模型中某一模块后，单击按钮



，断点/显示点列表将列出当前设置的显示点。在命令行模式下，该按钮对应的命令是 trace gcb 或 trace s:b。前者为当前模块设置显示点，后者可直接针对某个模块进行设置，其中 s 表示系统顺序，b 表示模块顺序。例如，在模型中选中 sum 模块，然后再命令行模式下调试，使用 next 命令执行到 TM=3 之后(即阶跃信号开始跳变)，执行 probe 命令，则可显示此时的 sum 模块输入/输出。

```
......
% ------------------------------------------------------------------------- %
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe
Entering block probe mode. Click on any block to see its data.
Type any command to leave probe mode.
Probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1          = [0.98177007864064314]
U2          = [0.018229921359356905]
Y1          = [0.98177007864064314]
.........
```

若要观察同一时刻其他模块的输入/输出，可在模型中选中相应的模块，命令行中会立即显示其当前输入/输出。例如选中 PID 模块，命令行输出以下信息：

```
...........
% ------------------------------------------------------------------------- %
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe
Entering block probe mode. Click on any block to see its data.
Type any command to leave probe mode.
Probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1          = [0.98177007864064314]
U2          = [0.018229921359356905]
Y1          = [0.98177007864064314]
Probe: Data of SubSystem block (virtual) 'DC_MOTOR_PID/PID Controller':
U1          = [0.98177007864064314]
Y1          = [8.6319002707974715]
...........
```

(2) 显示选中模块的输入/输出。在 GUI 界面下，选中模型中某一模块后，单击按钮会在 Output 界面中显示当前状态下选中模块的其 I/O 信息，在命令行模式下，probe s:b 命令与之对应。

选中 sum 模块并单击按钮后可得到图 2.3.16 所示的输出结果。

在命令行模式下，选中分别选中 PID 模块和 sum 模块并执行 probe gcb 指令，命令窗口中结果如下：

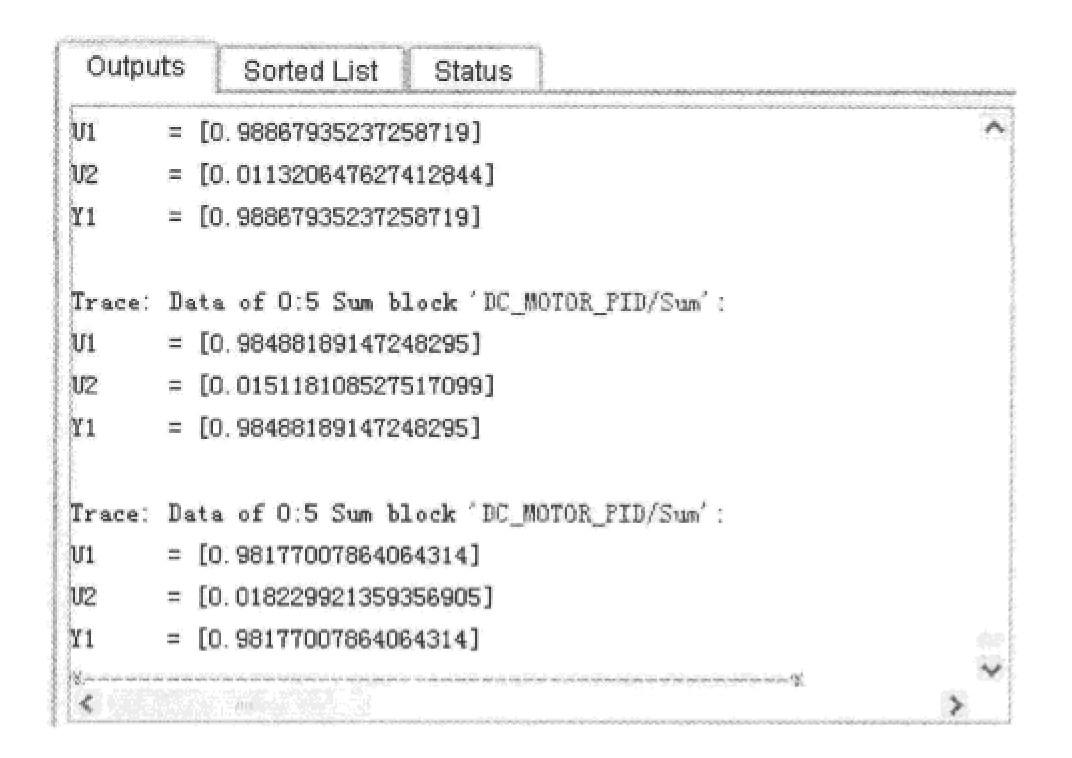


图 2.3.16 outputs 界面

```
............
% ----------------------------------------------------------------- %
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> probe gcb
probe: Data of SubSystem block (virtual) 'DC_MOTOR_PID/PID Controller':
U1        = [0.98177007864064314]
Y1        = [8.6319002707974715]

(sldebug @41): >> probe gcb
probe: Data of 0:5 Sum block 'DC_MOTOR_PID/Sum':
U1        = [0.98177007864064314]
U2        = [0.018229921359356905]
Y1        = [0.98177007864064314]

(sldebug @41): >>
.......
```

使用 probe 指令可以使调试器进入 probe 状态，这时用户选中模型的任意一个非虚模块时，命令窗口中就会显示其 I/O 信息，这比上面的方法更加简单。再次输入 probe 指令可退出该模式。

(3) 显示断点处的模块输入/输出信息。在命令行模式下，调试命令 disp gcb / disp s:b 用于设置当前或指定的模块在调试中断时显示其输入/输出，命令 undisp gcb / undisp s:b 可移去当前或指定的显示点(其中 s 表示系统顺序，b 表示模块顺序)。

该指令只能在命令行模式下运行，在 GUI 界面中无法实现。

在设置中断点后，每单步调试一个模块，命令行窗口都会显示一次该模块的输入/输出。例如，输入指令 disp 0:0，将会显示模块：

```
DC_MOTOR_PID/DC_motor_subsystem/Integrator 的输入/输出信息：
...........
```



```
% ---------------------------------------------------------------------- %
[TM = 3.020454598515591      ] DC_MOTOR_PID.Outputs.Major
(sldebug @41): >> disp 0:0
Installed data display of 0:0 Integrator block '
DC_MOTOR_PID/DC_motor_subsystem/Integrator'.

(sldebug @41): >> next

Disp: Data of 0:0 Integrator block '
DC_MOTOR_PID/DC_motor_subsystem/Integrator':
U1        = [17.263800541594943]
Y1        = [0.033780793929875291]
CSTATE    = [0.37426778201746674]
% ---------------------------------------------------------------------- %
[TM = 3.020454598515591      ] DC_MOTOR_PID.Update
(sldebug @65): >>
......
```

注意,0:0 为 DC_MOTOR_PID/DC_motor_subsystem/Integrator 模块的 ID 序号,每个模块的 ID 序号都可以在 Sorted List 子界面中找到,下面将具体介绍该界面。

2. 显示模型信息

(1) 显示模块执行顺序。GUI 界面的 Sorted List 选项卡显示了模型中的每个模块和非虚子系统。该界面列出了各模块的执行顺序及模块 ID 号,如图 2.3.17 所示。



图 2.3.17　Sorted List 选项卡

在命令行模式下可以用 slist 命令来显示模型的列表顺序:

```
(sldebug @0): >> sldebug'DC_motor_PID'
```

```
(sldebug @0): >> slist
(sldebug @0): >> slist

--- Sorted list for 'DC_MOTOR_PID' [22 nonvirtual blocks, directFeed = 0]
  0:0    'DC_MOTOR_PID/DC_motor_subsystem/Integrator' (Integrator)
  0:1    'DC_MOTOR_PID/DC_motor_subsystem/Integrator2' (Integrator)
  0:2    'DC_MOTOR_PID/DC_motor_subsystem/Integrator1' (Integrator)
  0:3    'DC_MOTOR_PID/Signal Builder/FromWs' (FromWorkspace)
  0:4    'DC_MOTOR_PID/Scope' (Scope)
  0:5    'DC_MOTOR_PID/Sum' (Sum)
  0:6    'DC_MOTOR_PID/PID Controller/Proportional Gain' (Gain)
  0:7    'DC_MOTOR_PID/PID Controller/Integrator' (Integrator)
  0:8    'DC_MOTOR_PID/PID Controller/Derivative Gain' (Gain)
  0:9    'DC_MOTOR_PID/PID Controller/Filter' (Integrator)
  0:10   'DC_MOTOR_PID/PID Controller/SumD' (Sum)
  0:11   'DC_MOTOR_PID/PID Controller/Filter Coefficient' (Gain)
  0:12   'DC_MOTOR_PID/PID Controller/Sum' (Sum)
  0:13   'DC_MOTOR_PID/DC_motor_subsystem/Gain5' (Gain)
  0:14   'DC_MOTOR_PID/DC_motor_subsystem/Gain1' (Gain)
  0:15   'DC_MOTOR_PID/DC_motor_subsystem/Add' (Sum)
  0:16   'DC_MOTOR_PID/DC_motor_subsystem/Gain2' (Gain)
  0:17   'DC_MOTOR_PID/DC_motor_subsystem/Gain4' (Gain)
  0:18   'DC_MOTOR_PID/DC_motor_subsystem/Add1' (Sum)
  0:19   'DC_MOTOR_PID/DC_motor_subsystem/Gain' (Gain)
  0:20   'DC_MOTOR_PID/DC_motor_subsystem/Gain3' (Gain)
  0:21   'DC_MOTOR_PID/PID Controller/Integral Gain' (Gain)
```

(2) 显示调试器状态。在 GUI 界面下，Status 选项卡中列出了调试器各种选项的设置情况，例如条件断点，如图 2.3.18 所示。

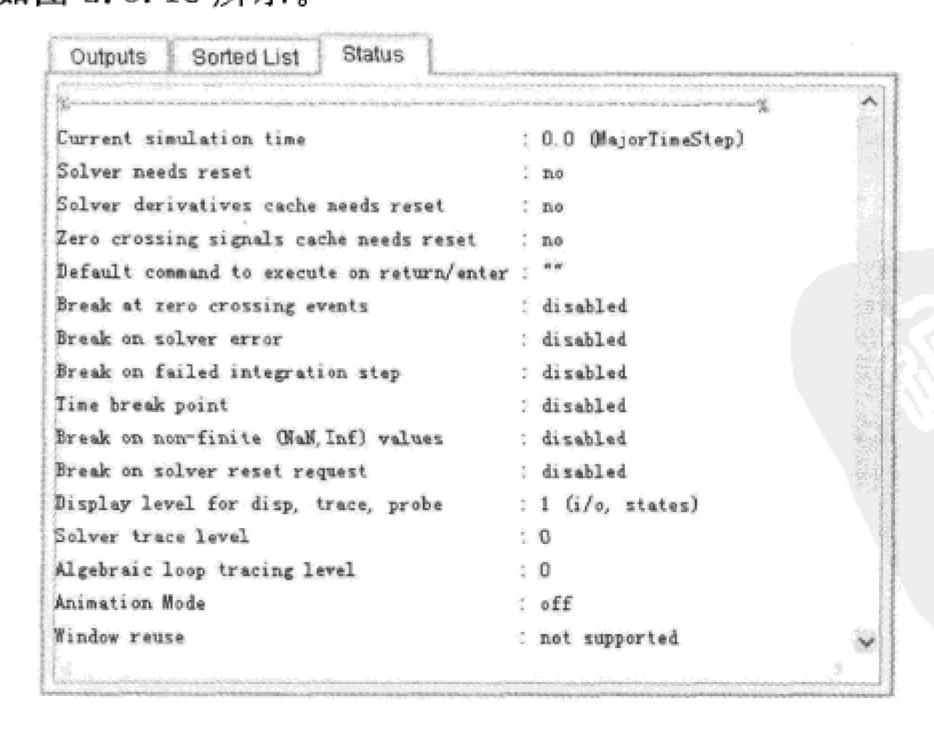
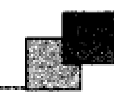


图 2.3.18 Status 选项卡



在命令行模式下，可以使用 status 指令来显示调试器的设置情况：

```
% ---------------------------------------------------------------------- %
[TM = 0                 ] DC_MOTOR_PID.Simulate
(sldebug @0): >> status
% ---------------------------------------------------------------------- %
Current simulation time                              : 0.0 (MajorTimeStep)
Solver needs reset                                   : no
Solver derivatives cache needs reset                 : no
Zero crossing signals cache needs reset              : no
Default command to execute on return/enter : ""
Break at zero crossing events                        : disabled
Break on solver error                                : disabled
Break on failed integration step                     : disabled
Time break point                                     : disabled
Break on non-finite (NaN,Inf) values                 : disabled
Break on solver reset request                        : disabled
Display level for disp, trace, probe                 : 1 (i/o, states)
Solver trace level                                   : 0
Algebraic loop tracing level                         : 0
Animation Mode                                       : off
Window reuse                                         : not supported
Execution Mode                                       : Normal
Display level for etrace                             : 0 (disabled)
Break points                                         : none installed
Display points                                       : none installed
Trace points                                         : none installed
```

(3) 显示潜在过零点模块。在仿真过程中 zclist 指令能够列出模型中所有可能会产生非采样过零点的模块：

```
% ---------------------------------------------------------------------- %
[TM = 0                 ] DC_MOTOR_PID.Simulate
(sldebug @0): >> zclist
   0 0:3:0     R  FromWorkspace'DC_MOTOR_PID/Signal Builder/FromWs'
```

第 3 章

Stateflow 建模与应用

Stateflow 是有限状态机(finite state machine)的图形工具,它通过开发有限状态机和流程图扩展了 Simulink 的功能。Stateflow 使用自然、可读和易理解的形式,使复杂的逻辑问题变得清晰与简单,并且与 MATLAB/Simulink 紧密结合,为包含控制、优先级管理、工作模式逻辑的嵌入式系统设计提供了有效的开发手段,是本书的核心内容之一。读者在第 5～8 章将看到 Stateflow 应用于 MCU 器件的嵌入式开发,一些采用传统方法难以实现的算法,利用 Stateflow 建模将显得非常容易。利用 Stateflow Coder 代码生成工具,可以将 Stateflow 状态图模型直接生成 C 代码。

Stateflow 的主要功能包括以下几方面:

(1) 使用层次化、可并行的、有明确执行语义的元素,来描述复杂的逻辑系统。

(2) 采用流程图定义图形化函数。

(3) 利用真值表实现表格形式的功能。

(4) 使用临时逻辑处理状态转移与事件。

(5) 支持 Mealy 和 Moore 有限状态机。

(6) 可集成用户自定义的 C 代码。

(7) 可用动画的形式显示状态图的仿真运行过程,并可记录数据。

(8) 调试器使用图形化断点进行单步调试,并可观察其中的数据。

本章主要内容:

(1) Stateflow 工作原理与基本概念。

(2) 建立 Stateflow 状态图与流程图。

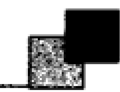
(3) Stateflow 的层次结构与并行机制。

(4) Stateflow 应用。

3.1 Stateflow 基本概念

Stateflow 对象可分为图形对象与非图形对象。

图形对象有状态、历史节点、迁移、默认迁移、连接节点、真值表、图形函数、Embedded



MATLAB 函数、盒函数、Simulink 函数；非图形对象有事件、数据、目标。本节首先介绍常用的对象：状态、迁移、数据、事件的概念和使用。

Stateflow 状态机使用一种基于容器的层次结构管理 Stateflow 对象，也就是说，一个 Stateflow 对象可以包含其他 Stateflow 对象。

最高级的对象是 Stateflow 状态机，它包含了所有的 Stateflow 对象，因此也就包含了 Simulink 中的所有 Stateflow 状态图，以及数据、事件、目标对象。

同样地，状态图包含了状态、盒函数、函数、数据、事件、迁移、节点与注释事件(note events)。用户可以使用这一系列对象，建立一个 Stateflow 状态图。而具体到一个状态，它也可以包含上述的对象。

图 3.1.1 抽象地说明了这样的关系，而图 3.1.2 则具体地说明了 Stateflow 状态机的组成。

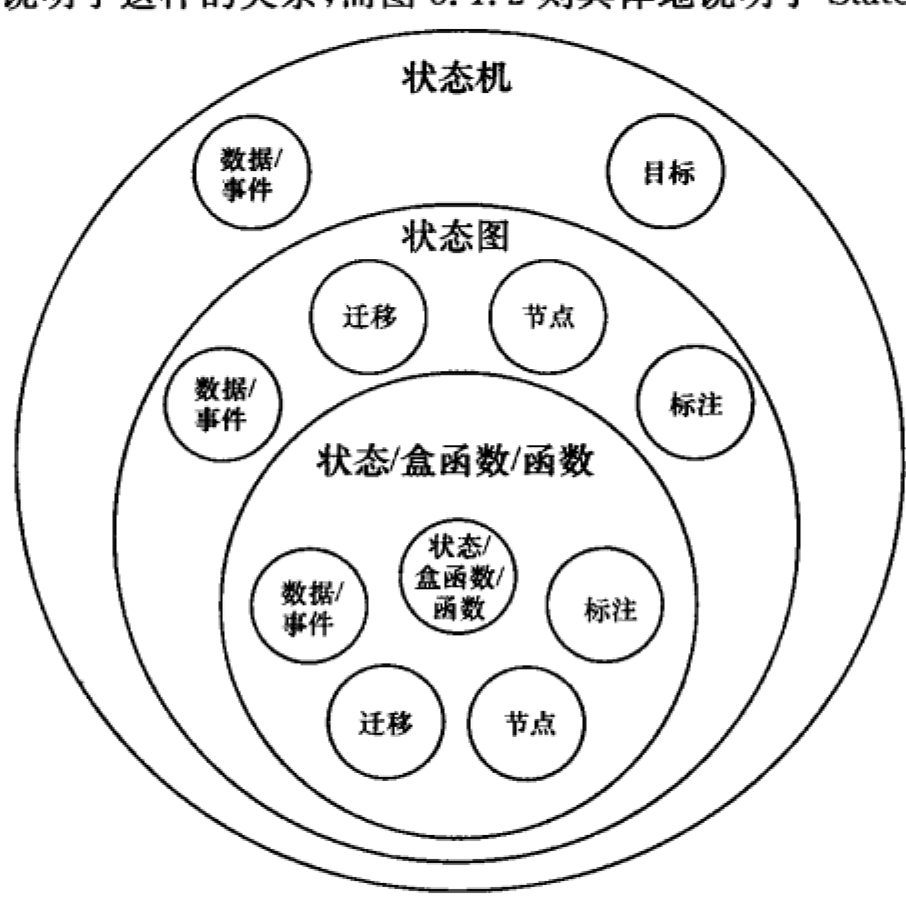


图 3.1.1 Stateflow 层次结构(数据字典)

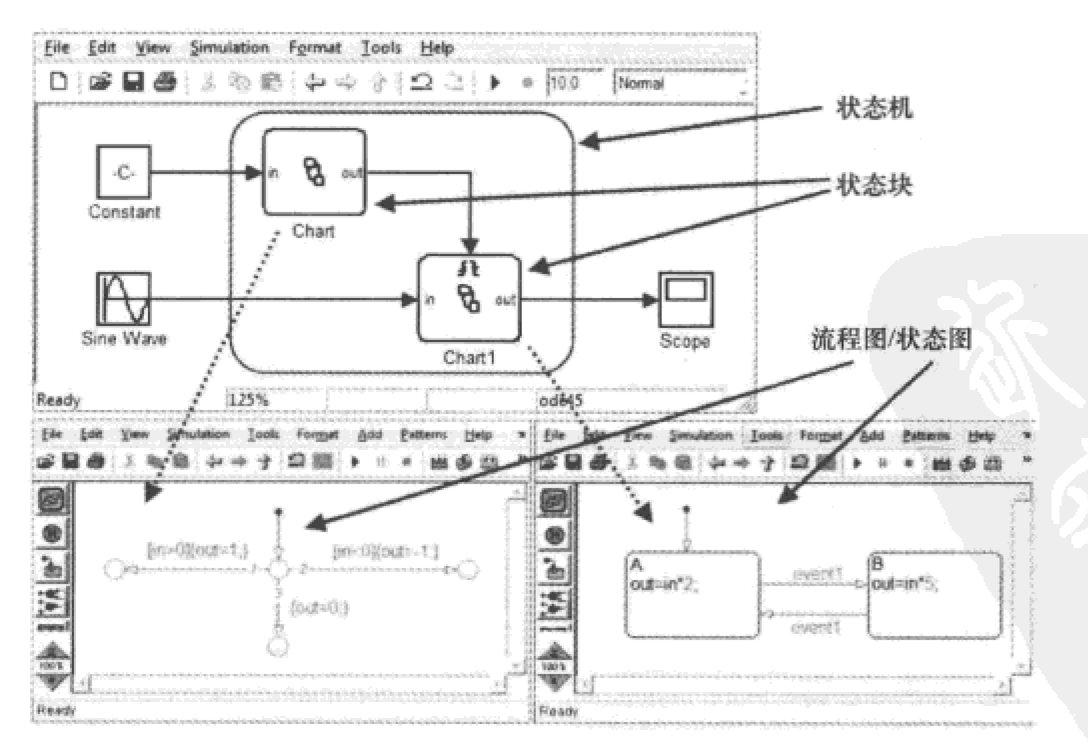


图 3.1.2 Stateflow 状态机的组成

3.1.1 状态图编辑器

在 Simulink 模块库浏览器中，找到 Stateflow 模块，如图 3.1.3 所示。添加入模型窗口，如图 3.1.4 所示。

用户也可以使用以下命令，建立带有 Stateflow 状态图的 Simulink 模型，如图 3.1.4 所示。同时打开 Stateflow 模块库，如图 3.1.5 所示。

```
>> sf
```

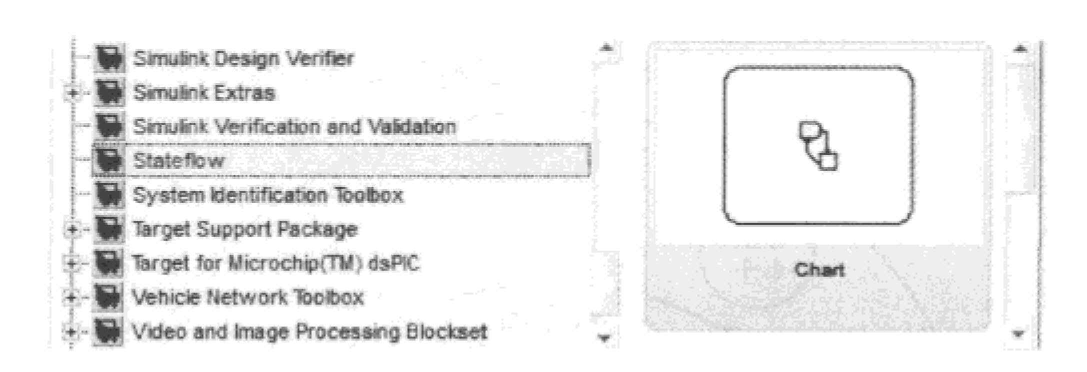
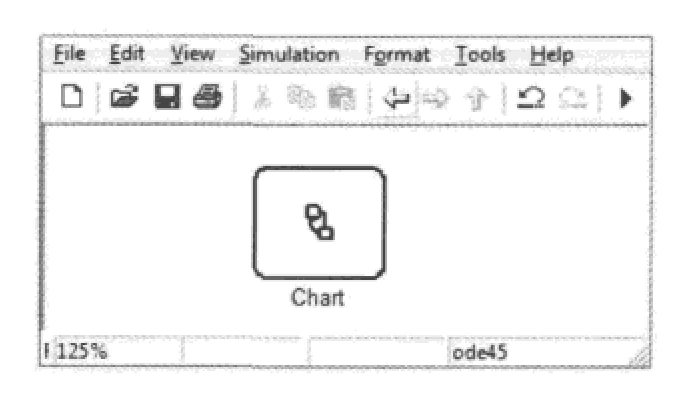


图 3.1.3　Stateflow 模块



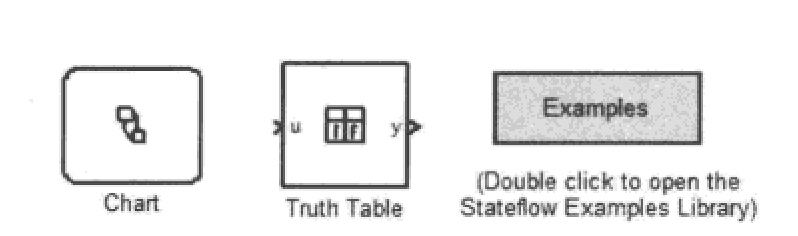


图 3.1.4　带有 Stateflow 状态图的 Simulink 模型　　　　图 3.1.5　Stateflow 模块库

用户还可以直接使用以下命令，快速建立带有 Stateflow 状态图的 Simulink 模型。

```
>> sfnew
```

双击 Chart 模块，打开 Stateflow 编辑器窗口，如图 3.1.6 所示，左侧工具栏列出了 Stateflow 图形对象的按钮。

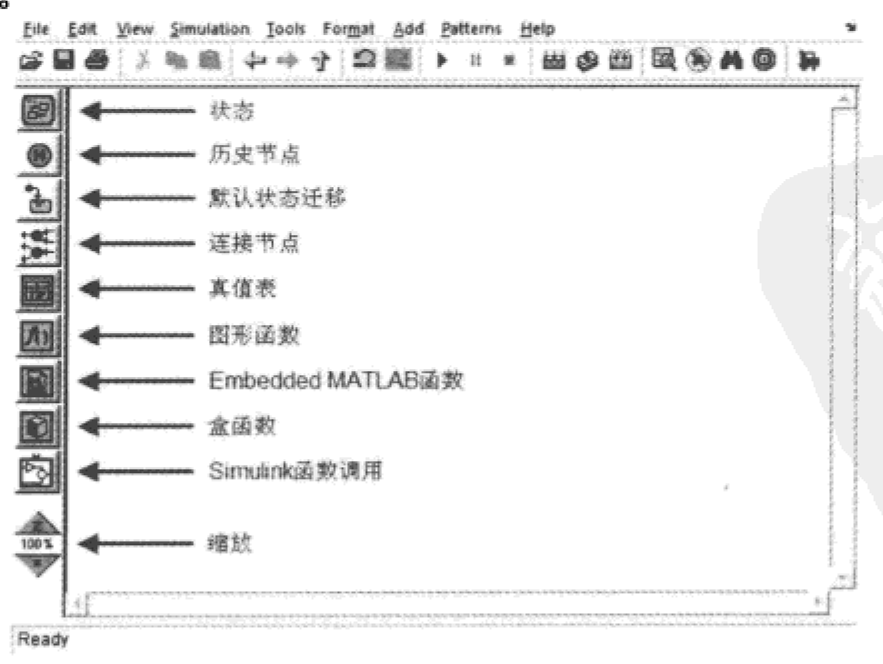
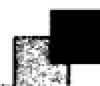


图 3.1.6　Stateflow 编辑器窗口



3.1.2 状　态

状态可以理解为事件驱动系统中的模式，可分为激活与非激活状态，而状态是否激活则是由状态图中的事件与条件来决定的，若没有预先定义的事件或条件发生，状态将一直保持其原先的激活或非激活状态。

1. 状态的层次结构

状态可以包含除了目标（详见第 3.6.6 节）以外的所有 Stateflow 对象，所以状态内部可以有其他状态，如图 3.1.7 所示，处于外层的 A 称作超状态（或父状态），处于内部的 B 称作子状态。

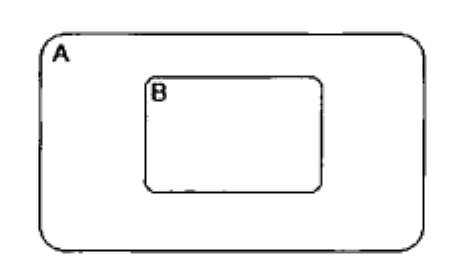


图 3.1.7　超状态与子状态

每一个状态都有其父状态，在图 3.1.7 中，状态 A 的父状态就是 Stateflow 状态图本身。

2. 状态的横向结构

在 Stateflow 状态图的顶层或某一超状态下，通常并存有多个状态，它们之间的关系可分为互斥与并行。

(1) 互斥状态（OR）。互斥状态的矩形框边缘显示为实线，同一级的互斥状态，至多允许激活一个状态。互斥状态如图 3.1.8 所示，状态 A 与状态 B 是互斥的，它们只能有一个处于激活状态；当状态 A 被激活时，同样其子状态 A1 与 A2 也只能有一个处于激活状态。

(2) 并行状态（AND）。并行状态的矩形框边缘显示为虚线，同一级的并行状态，可在同一时刻被激活。并行状态如图 3.1.9 所示。状态 A 与状态 B 是并行的，它们可同时处于激活状态；子状态 A1 与 A2 也同时处于激活状态，而子状态 B1 与 B2 只能有一个处于激活状态。

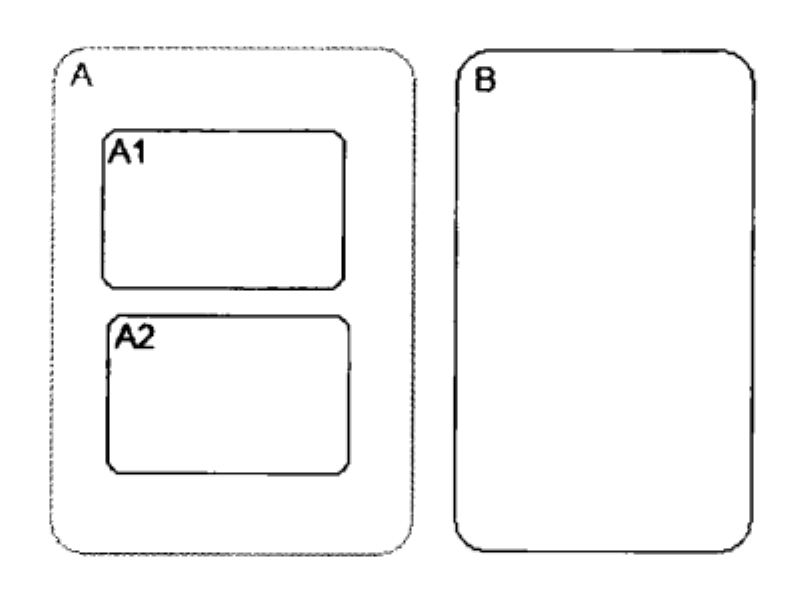


图 3.1.8　互斥状态

图 3.1.9　并行状态

状态层次结构与并行机制的详细概念与应用，见 3.4 节与 3.5 节。

3. 状态标签

状态名仅是状态标签的一部分，完整的标签格式如下，第一行是状态名，以下若干行是各类动作，用户可以设置全部或部分的状态动作，当然也可以不设置任何动作。

```
name/                                          % 状态名
entry:        entry actions                    % 进入该状态时的动作
during:       during actions                   % 处于该状态时的动作
exit:         exit actions                     % 退出该状态时的动作
on event_name:   on event_name actions         % 某事件发生时的动作
bind:         events, data                     % 指定需要限制作用范围的事件与数据
```

(1) 状态名。状态名可由字母、数字、下画线组成，如果状态名后跟随的是回车符，则斜线是可有可无的。根据 Stateflow 的分层结构，同级的各个子状态不允许重名，但不同级的状态则不受限制。

图 3.1.10 所示的 Stateflow 状态图是有效的，尽管看上去状态 On、Off 有重名现象，但在 Stateflow 分层结构中，它们的全名分别如下：



```
A.On
A.Off
B.On
B.Off
```

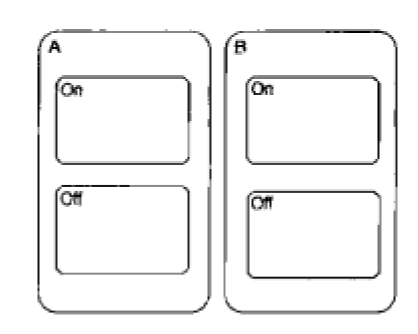


图 3.1.10　状态名

(2) 状态动作。状态动作如表 3.1.1 所列。

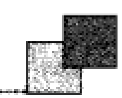
表 3.1.1　状态动作类型

动作类型	缩写	说　明
entry	en	进入当前状态时的动作
during	du	处于当前状态，并且某事件发生时的动作
exit	ex	离开当前状态时的动作
bind	无	约束一个事件或数据，使得仅当前状态及其子状态有权限广播该事件或修改该数据
on event_name	无	当前状态接收 1 次广播事件时的动作
on after(n,event_name)	无	当前状态完整接收 n 次广播事件后的动作
on before(n,event_name)	无	当前状态完整接收 n 次广播事件前的动作
on at(n, event_name)	无	当前状态完整接收 n 次广播事件时的动作
on every(n,event_name)	无	当前状态每接收 n 次广播事件时的动作

每个动作类型，用户可指定多个具体动作，每个动作之间以回车、分号、逗号区隔，动作类型关键词后必须跟随一个半角冒号。

① entry 动作。关键词为 entry(或缩写为 en)。如果用户在状态名后加入斜线，并直接跟随具体动作，则该动作默认为进入动作。如图 3.1.11 所示，进入状态 A 时，y=3，同时又执行 y++，最终的结果为 y=4。

② during 动作。关键词为 during(或缩写为 du)。如图 3.1.12 所示，进入状态 A 时，y=3，同时不断执行 y++。若求解器的定点步长取 0.2，仿真时长取 2，则最终的结果为 y=13。



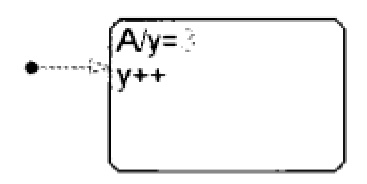
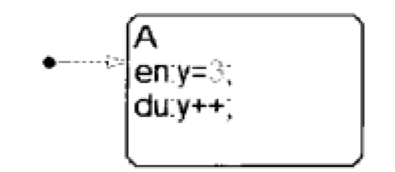


图 3.1.11 entry 动作



3.1.12 during 动作

③ exit 动作。关键词为 exit(或缩写为 ex)。如图 3.1.13 所示,系统处于状态 A,当 A 的激活时间达到 5 个仿真步长,退出状态 A,进入状态 B,最终的结果为 y=4,如图 3.1.14 所示。

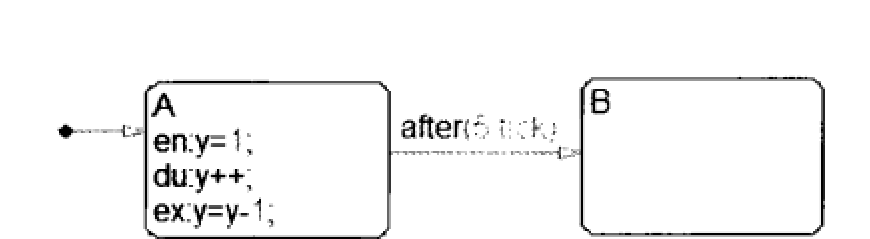


图 3.1.13 exit 动作

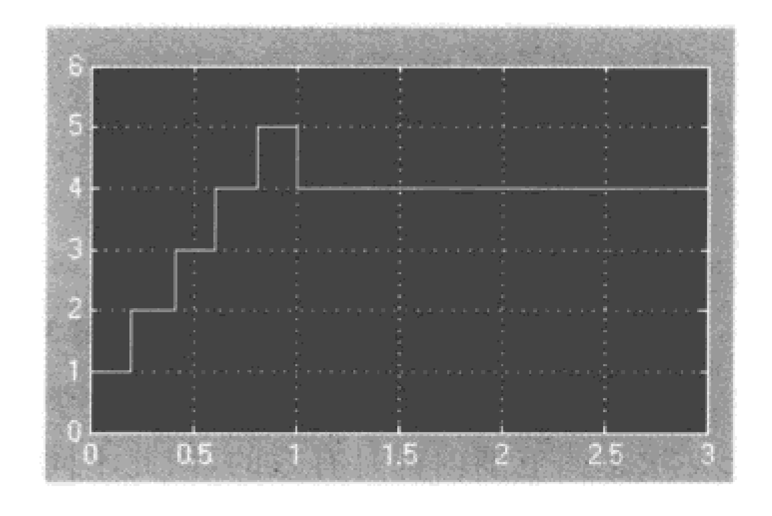


图 3.1.14 输出结果

④ 广播事件动作。表 3.1.1 所列的广播事件动作,能实现各种事件触发。

以单次广播事件动作为例,关键词为 on event_name,其中 event_name 表示某一广播事件名,事件名应是唯一的。如图 3.1.15 所示,系统处于状态 A,当检测到事件 stop,立即执行 c()。

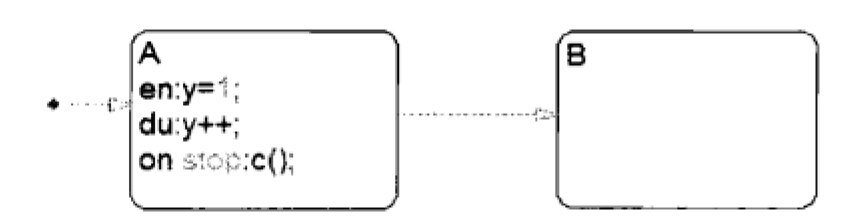


图 3.1.15 广播事件动作

⑤ bind 动作。关键词为 bind。如图 3.1.16 所示,变量 y、事件 start 被绑定在状态 A,这表示仅有 A 状态及其子状态有权限修改变量 y 并广播事件 start,其他状态 B 能够读取变量 y、监听到事件 start,但无权修改变量 y、广播事件 start。

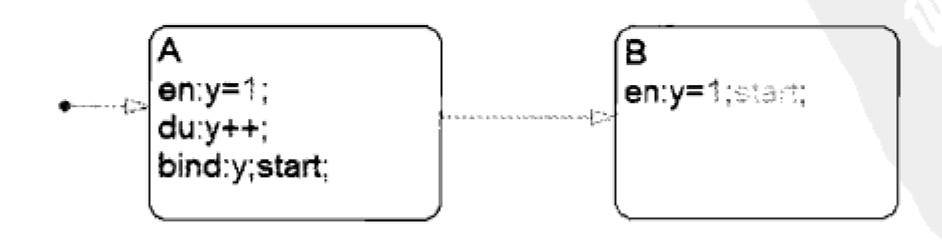


图 3.1.16 bind 动作

若运行该状态图,系统提示变量 y 仅能由状态 A 及其内部的状态迁移修改,事件 start 仅

能在状态 A 及其内部迁移进行广播，如图 3.1.17 所示。

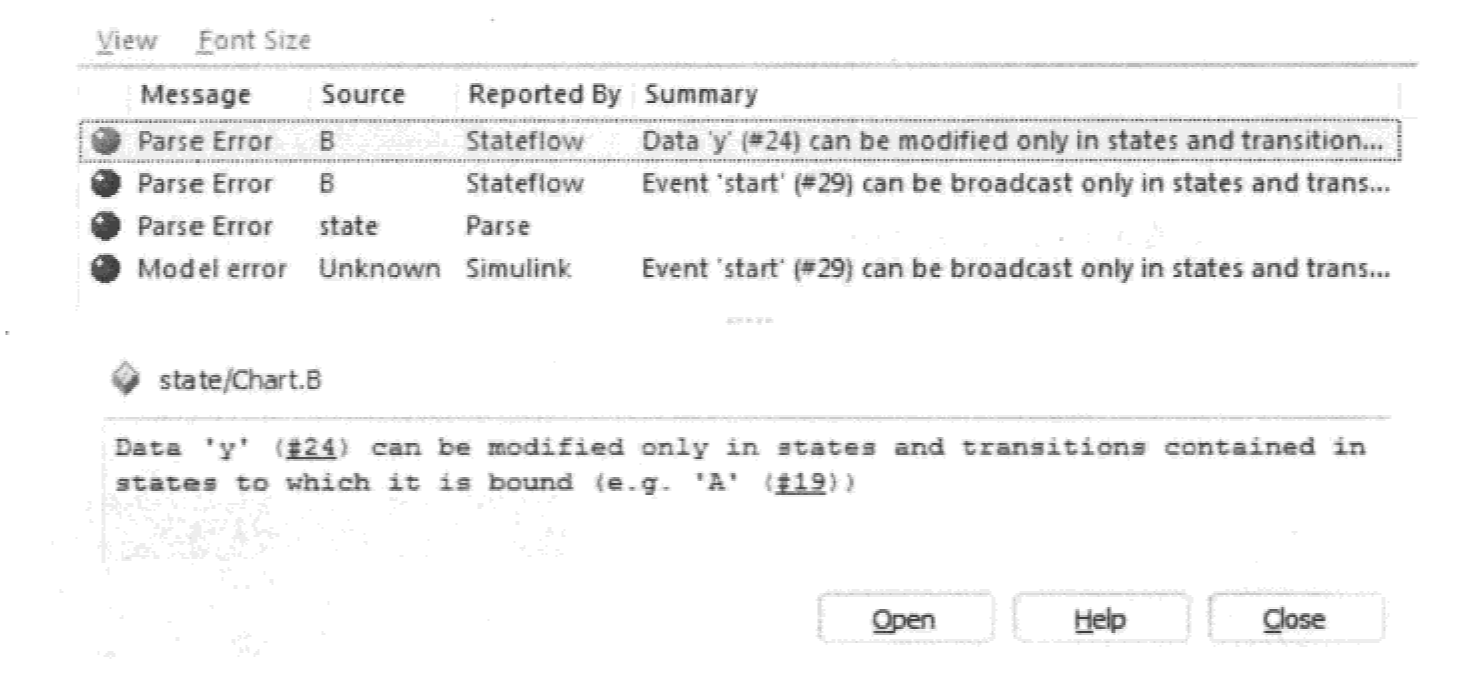


图 3.1.17 错误提示

与其他动作不同，bind 动作不需要判断当前状态是否已激活，也就是说它在整个 Stateflow 状态图范围内都是有效的，因此不同状态不允许约束同一个变量与事件。

如图 3.1.18 所示，状态 A、B 同时约束了变量 y，系统提示这是不允许的，如图 3.1.19 所示。

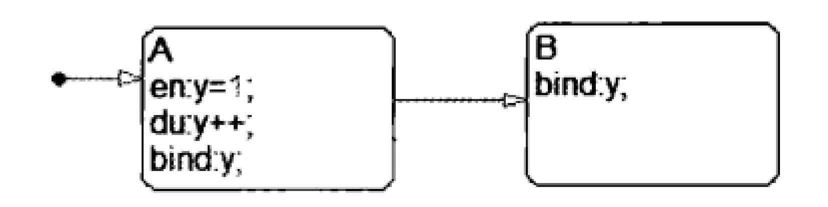


图 3.1.18 无效的 bind 动作

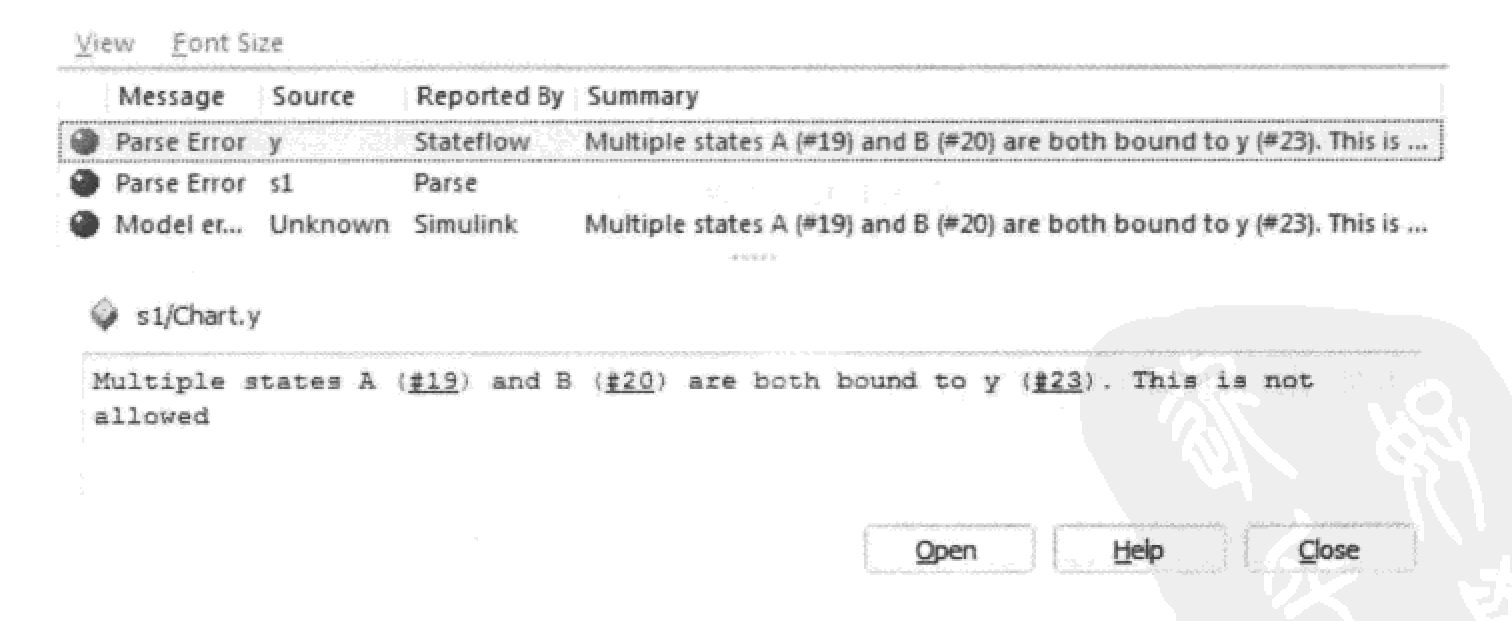
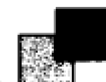


图 3.1.19 错误提示



3.1.3 迁 移

1. 迁 移

Stateflow 状态图使用一条单向箭头曲线表示迁移，它将两个图形对象连接起来，多数情况下，迁移是指系统从源状态向目标状态的转移。

在迁移曲线上加上标签，可以指定系统在何种条件下从源状态向目标状态转移。

如图 3.1.20 所示，当系统处于状态 A1 时间达到 1 s，即向状态 A2 迁移。

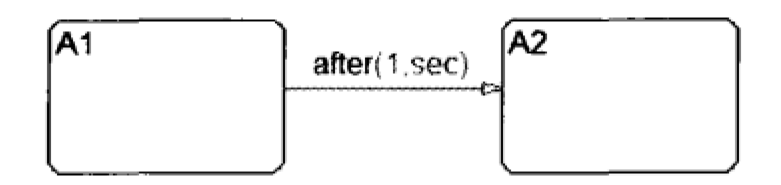


图 3.1.20 状态迁移

2. 默认迁移

默认迁移是一种特殊的迁移形式，它没有源对象。默认迁移用于指定同一级有多个互斥状态并存时，首先激活的状态。

某些情况下，默认迁移也可以加入标签，限制其所指向目标状态的激活。

如图 3.1.21 所示，状态 A1 与 A2 是互斥的，当它们的父状态 A 激活时，状态 A1 也同时激活。

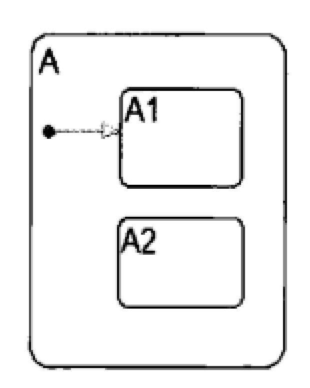


图 3.1.21 默认迁移

3. 迁移标签

迁移标签的完整格式如下，它可用于一般迁移与默认迁移，如图 3.1.22 所示。

```
event[condition]{condition_action}/transition_action
```

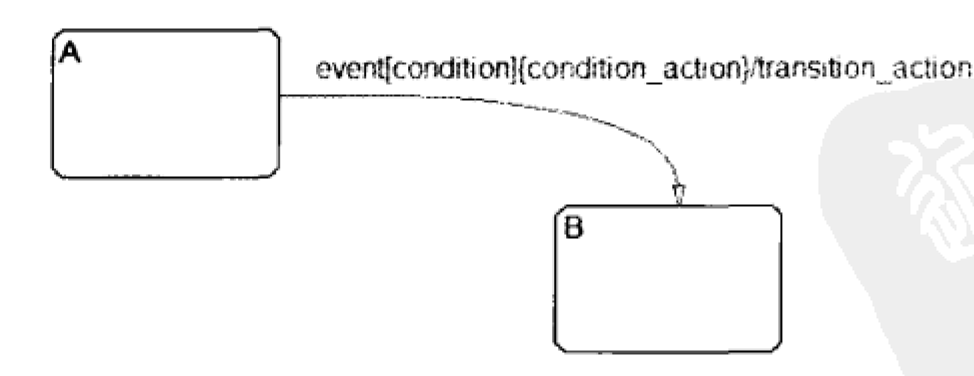


图 3.1.22 完整的迁移标签

各字段的意义如表 3.1.2 所列。

表 3.1.2 迁移标签字段

标签字段	说 明	标签字段	说 明
event	引发迁移的事件	[condition]	条件动作与迁移的发生条件
{condition_action}	当条件为真时，执行的动作	/transition_action	发生迁移，进入目标状态前所执行的动作

(1) 事件。事件是指定迁移的触发事件。如果用户另行指定了触发条件，则当条件为真，且发生该触发事件时，即发生迁移。这是个可选项，如果用户不指定触发事件，则任何事件都能够引发该迁移。多个触发事件之间使用逻辑或运算符"|"分隔。

如图 3.1.20 所示，当条件 after(1,sec)为真时，触发了迁移，系统状态从 A1 变成 A2。

(2) 条件。条件是一个布尔表达式，当它为真时，一旦发生指定的触发事件，则发生迁移。条件表达式的前后必须使用方括号"[ ]"包围。

如图 3.1.23 所示，当条件[y>=3]为真时，发生迁移。

(3) 条件动作。当条件表达式为真时，立刻执行条件动作。若事先未指定条件，系统则假设条件为真，并执行该条件动作。

如图 3.1.23 所示，当条件[y>=3]为真，条件动作{ y=10 }立刻执行。

(4) 迁移动作。当迁移目标有效时，执行迁移动作。若迁移标签由多个字段组成，则当整个标签有效时，执行迁移动作。

如图 3.1.23 所示，当条件[y>=3]为真，且目标状态 B 有效时，发生迁移，并执行迁移动作 z=20。运行结果如图 3.1.24 所示。

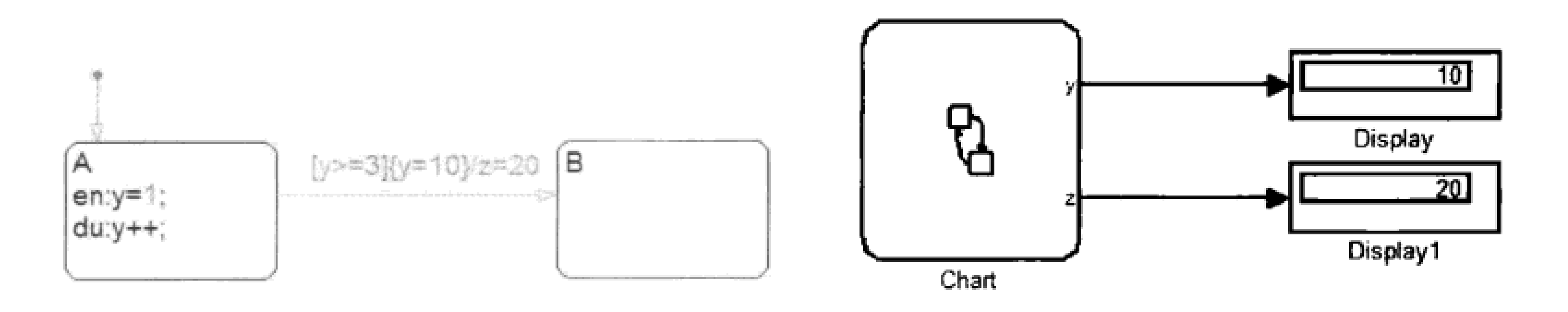


图 3.1.23 迁移条件与动作　　　　图 3.1.24 输出结果

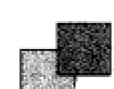
4. 迁移有效条件

对于非默认的迁移，当源对象处于激活状态且迁移标签有效时，发生迁移；对于默认迁移，当其父状态被激活时，发生迁移。

表 3.1.3 列出了迁移标签的有效条件，用户可以根据需要，选择性地输入迁移标签的部分或全部字段。

表 3.1.3 迁移标签有效条件

标签内容	标签有效条件	标签内容	标签有效条件
仅事件	该事件发生	事件与条件	该事件发生及条件为真
仅条件	任何事件发生及条件为真	仅行动	任何事件发生
空	任何事件发生		



3.1.4　数据与事件

图 3.1.1 所示的数据字典中，数据与事件是合并在一个圆圈内的，这表明它们有相似之处，本节对它们一起介绍。

1. 数　据

数据是非图形的对象，它有一个很重要的特性：作用域，即用户在使用数据时必须明确定义该特性。

根据作用域的不同，数据可分以下 8 种：

- Stateflow 状态图本地数据(Local)。
- 自外部 Simulink 模块输入的数据(Input from Simulink)。
- 向外部 Simulink 模块输出的数据(Output from Simulink)。
- 临时数据。
- 定义在 MATLAB 工作空间的数据。
- 常数(Constant)。
- 向 Simulink 模型与 Stateflow 状态图外部的目标(代码)导出的数据。
- 自 Simulink 模型与 Stateflow 状态图外部的源代码导入的数据。

数据的简单使用见 3.2.4 小节。

2. 事　件

事件也是非图形的对象，它驱动着整个 Stateflow 状态图的运行。如同数据，事件同样有它的作用域，根据作用域的不同，事件可分以下 3 种：

(1) Stateflow 状态图本地事件。

(2) 自外部 Simulink 模块输入的事件。

(3) 向外部 Simulink 模块输出的事件。

事件的简单使用见 3.2.4 节；事件的分类见 3.5 节。

3. 动　作

Stateflow 状态图支持状态动作、条件动作、迁移动作。这里所说的动作可以是一个函数调用、广播事件、数学运算等。

例如：

函数调用：ml. log10(x)；……

事件广播：Start；Stop；……

数学运算：x=1；y=2；z=x+y；……

3.1.5　对象的命名规则

以上简要介绍了常用对象的概念，用户可以使用任意的字母、数字与下画线的组合为这些对象命名，但名称不能以数字开头，中间也不能有空格。

由于 Real-Time Workshop 代码生成工具的限制，对象的名称不能超过一定长度，用户可以在模型参数设置对话框的 Real-Time Workshop→Symbols 面板中进行修改，默认的长度是 31，最大的长度是 256，如图 3.1.25 所示。

Auto-generated identifier naming rules

Maximum identifier length: 31

图 3.1.25 设置对象名称的长度

表 3.1.4 列出了一些关键字，它们是 Stateflow 动作语言的组成部分，因此是不能用来为对象命名的。

表 3.1.4 关键字

关键字	在 Stateflow 中的用途
hasChanged，hasChangedFrom，hasChangedTo	变更监测
complex，imag，real	复数数据
boolean，double，int8，int16，int32，single，uint8，uint16，uint32	数据类型
cast，fixdt，type	数据类型操作
send	明确事件
change，chg，tick，wakeup	隐含事件
false，inf，true，t	标志位
matlab，ml	MATLAB 函数与数据
bind，du，during，en，entry，ex，exit，on	状态动作
in	状态激活
after，at，before，every，sec，temporalCount	时间逻辑

3.2 Stateflow 状态图

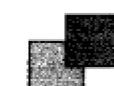
长跑比赛时，通常要用到以圈计时的方法，它的意思是：计时器初次开启时，两组数码管皆清零；运动员出发时，按 Start 按钮开始计时，数码管 1 显示实时时间；第一次回到起点，表示跑完一圈，这时按 LAP 按钮，数码管 2 显示当前的时间值，表示一圈所花费的时间，但比赛仍在进行，因此计时器仍然在计时；再次按 Start 按钮，两组数码管还是显示最后的时间；第三次按 Start 按钮，两组数码管清零，回到初始状态。本节以此为例，说明 Stateflow 状态图的建立过程。

3.2.1 状　态

1. 添加状态

新建一个空白的 Stateflow 模型，单击"状态"按钮，并在 Stateflow 窗口的适当位置再次单击，加入一个状态，如图 3.2.1 所示。

在加入之前，用户可随时按 ESC 键，或再次单击按钮，取消添加。



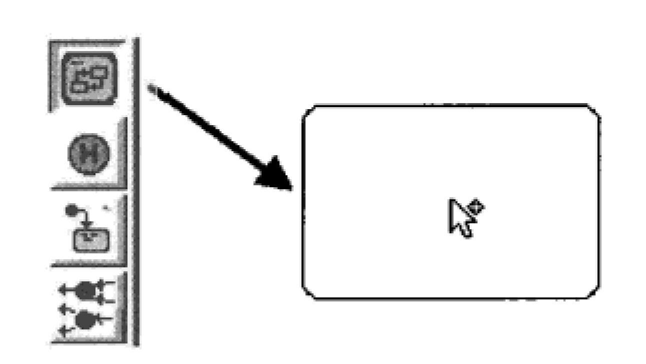


图 3.2.1 添加状态

2. 状态命名

在状态矩形框左上角的编辑提示符后，输入状态的名称，如 stop，如图 3.2.2 所示。若需要修改状态名，可将鼠标指针移至名称附近，待鼠标指针变成编辑样式时，再单击按钮进行修改，如图 3.2.3 所示。



图 3.2.2 状态命名　　图 3.2.3 状态名修改

3. 添加子状态

将鼠标指针移至状态矩形框 4 个角落的任意一个，调整其大小，如图 3.2.4 所示。

再按步骤 1、2，添加状态 Reset、Finished，放置在状态 Stop 的矩形框内，这时 Stop 为父状态，Reset、Finished 为子状态，如图 3.2.5 所示。

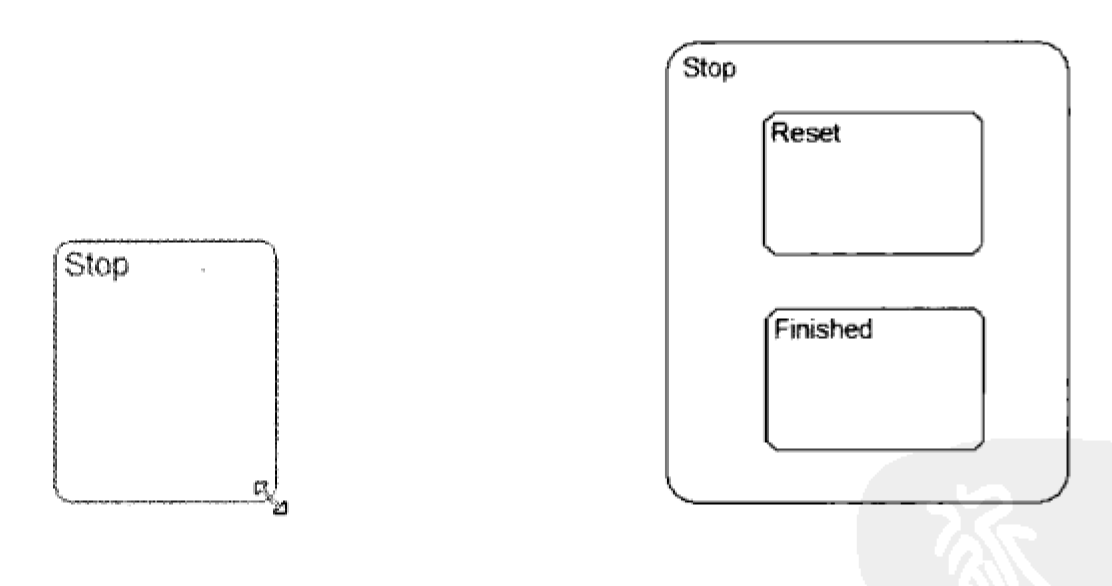


图 3.2.4 调整状态框　　图 3.2.5 父状态与子状态

3.2.2 迁 移

1. 添加迁移

将鼠标指针移至源状态矩形框的边缘，当鼠标指针变成十字时，如图 3.2.6 所示，按住鼠

标左键并拖向目标状态的边缘，然后释放，如图 3.2.7 所示，即添加了一个迁移。

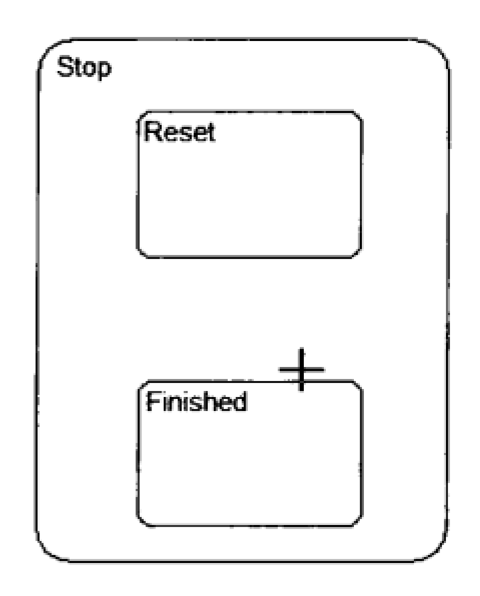



图 3.2.6 迁移起点

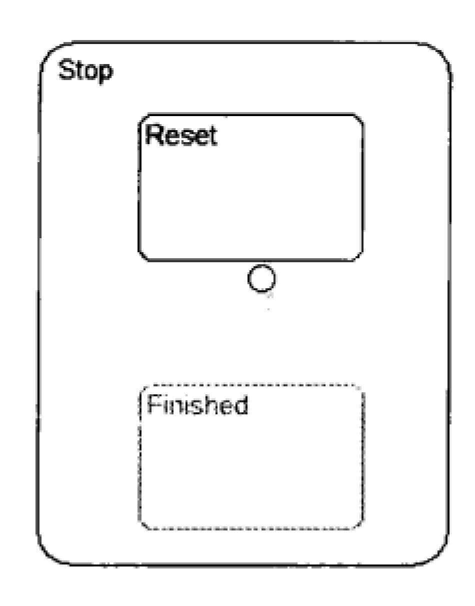


图 3.2.7 迁移终点

2. 添加默认迁移

单击按钮，将鼠标指针移至默认状态矩形框的水平或垂直边缘，如图 3.2.8 所示。

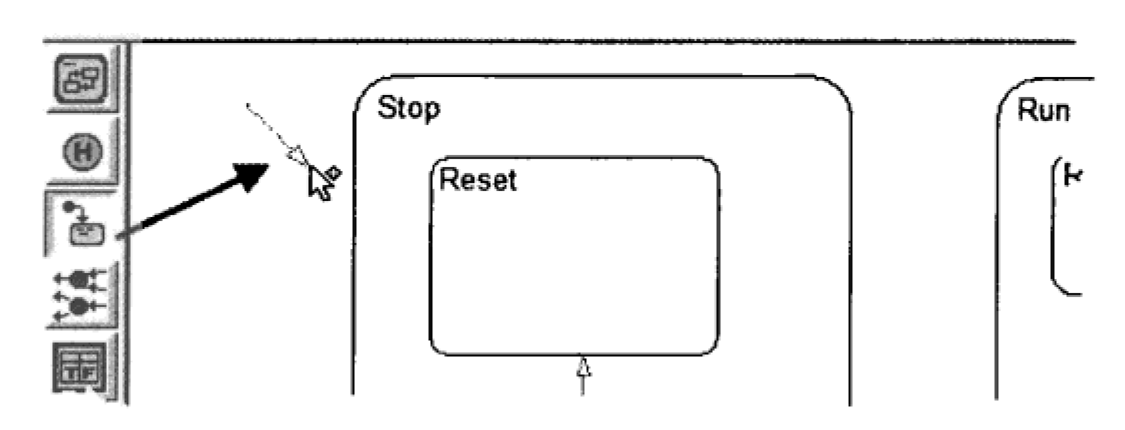


图 3.2.8 选择默认迁移

再次单击，即添加了一个默认迁移，如图 3.2.9 所示。

由于 Stop 是父状态，还需要针对其中的子状态，设置默认迁移，如图 3.2.10 所示，Stateflow 的层次结构详见 3.4 节。

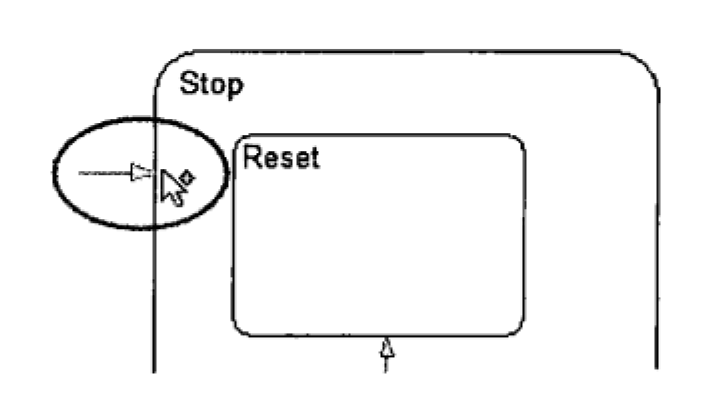


图 3.2.9 添加默认迁移

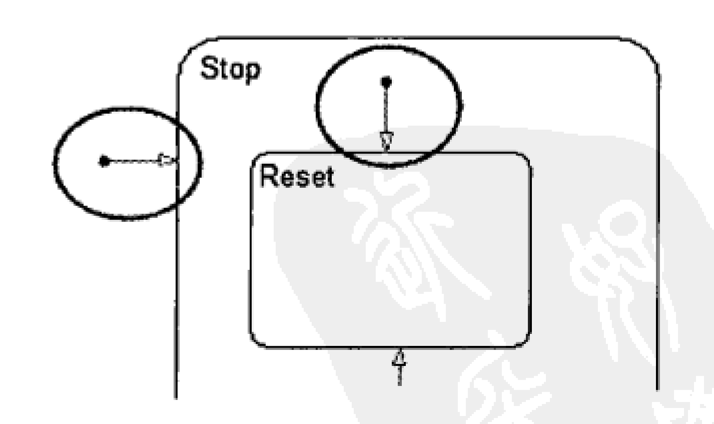
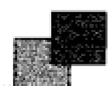


图 3.2.10 添加子状态默认迁移

3. 迁移变更

鼠标指针放置在迁移的起点或终点，当鼠标指针变成圆圈时（图 3.2.11），按住鼠标左键，



可将该端点移至其他状态，如图 3.2.12 所示。

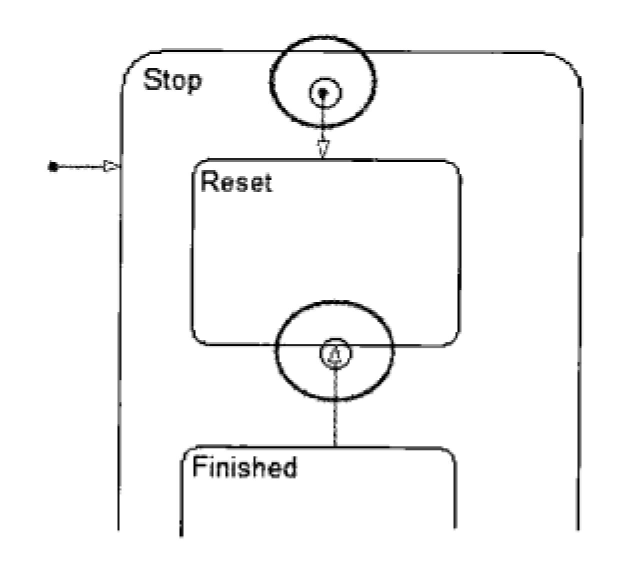


图 3.2.11　开始迁移变更

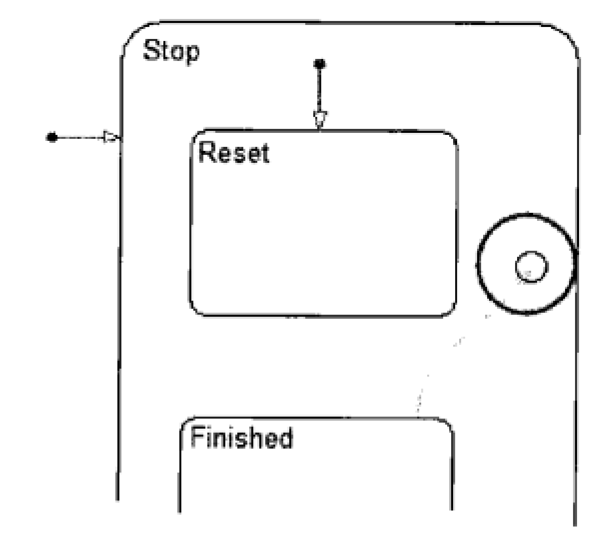


图 3.2.12　完成迁移变更

将默认迁移的起点移至某一状态，即转换为一般迁移；将一般迁移的起点悬空，即转化为默认迁移，如图 3.2.13 所示。若迁移终点悬空，则该迁移无效，如图 3.2.14 所示。

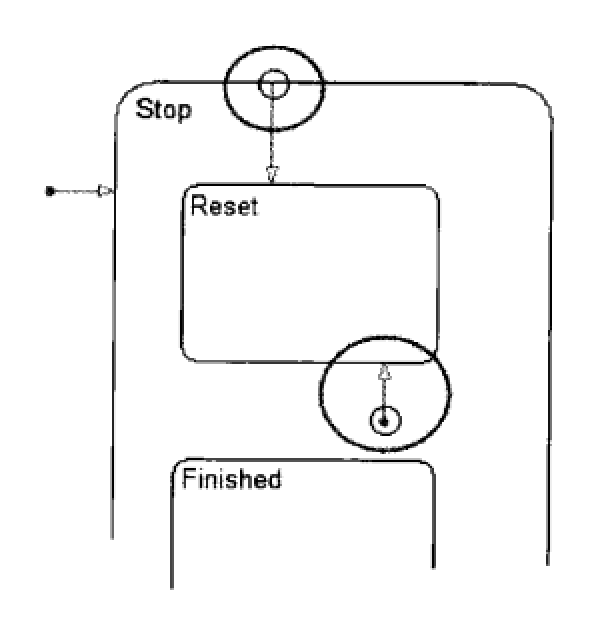




图 3.2.13　迁移起点变更

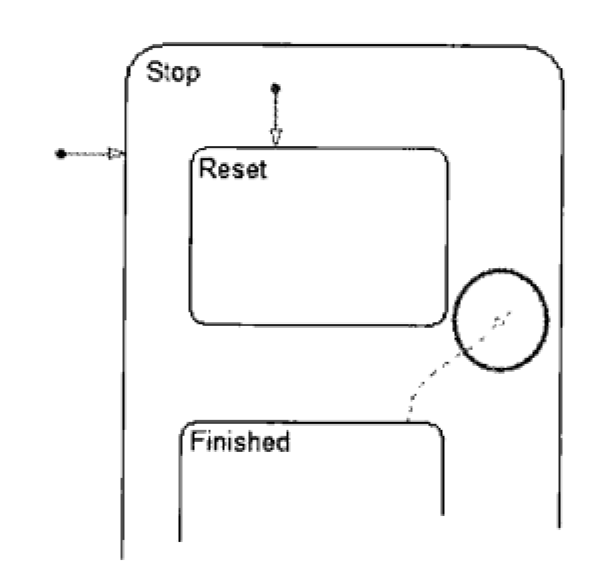


图 3.2.14　迁移终点变更

4. 迁移标签

新建的迁移标签不包含任何文字信息，用户单击迁移曲线一次，曲线上方显示“?”，如图 3.2.15 所示。

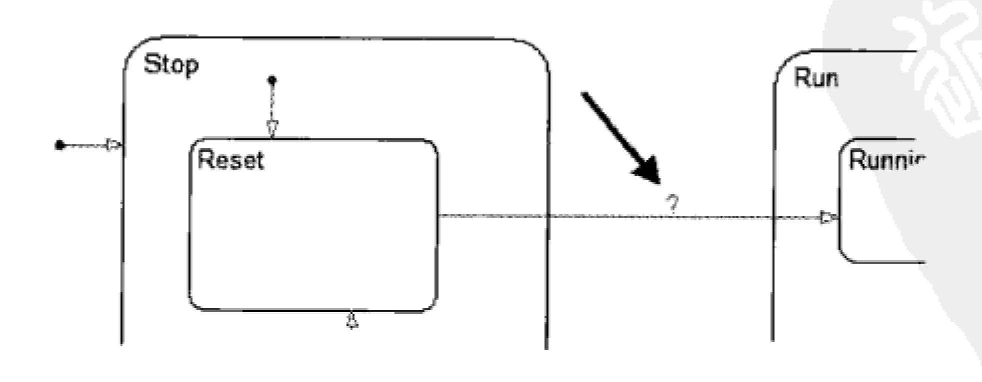
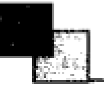


图 3.2.15　添加迁移标签

将鼠标指针移至“?”附近，再次单击，当显示编辑光标时，可编辑迁移标签，如图 3.2.16



所示。

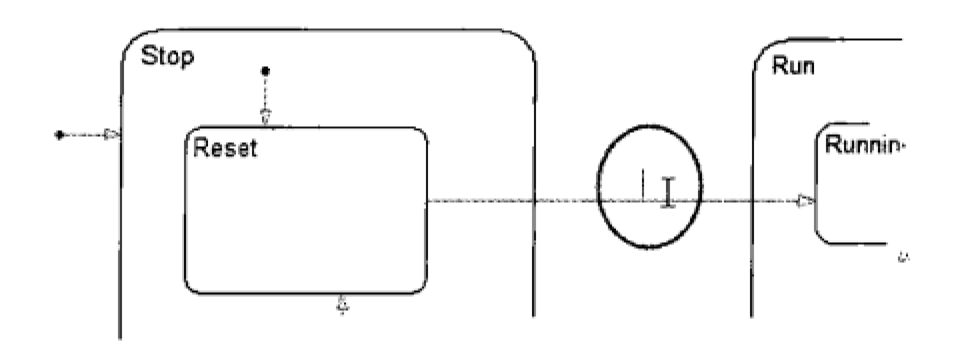


图 3.2.16 编辑迁移标签

完成编辑后，将鼠标指针放在标签的任意位置，按住鼠标左键并拖动，调整标签的位置，如图 3.2.17 所示。

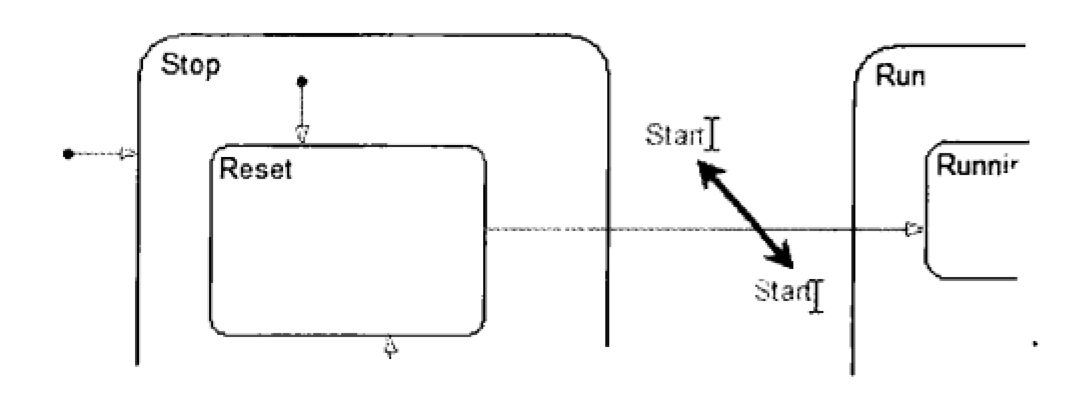


图 3.2.17 移动迁移标签

3.2.3 计时器状态图

根据以圈计时的特点，整个系统可分为 2 个父状态：停止与运行。停止状态包含 2 个子状态：计时器清零 Reset、计时器停止 Finished；运行状态也包含 2 个子状态：计时器运行 Running、以圈计时 LAP，如图 3.2.18 所示。

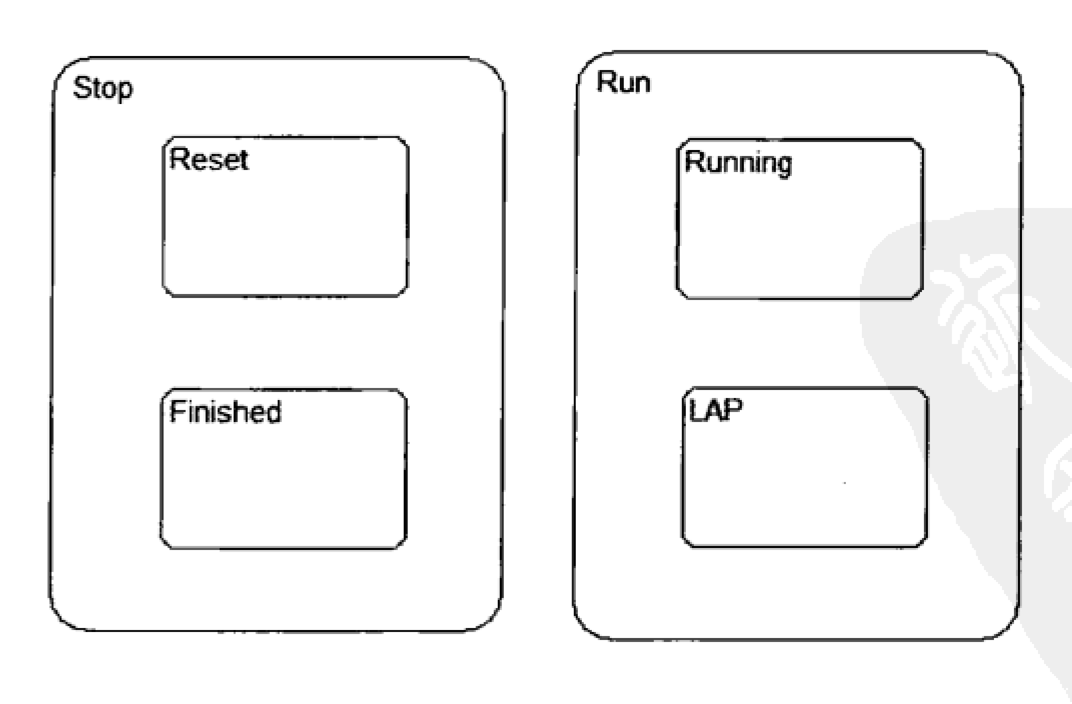
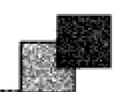


图 3.2.18 添加 4 个状态



再根据各状态之间的联系，添加默认迁移、迁移以及迁移标签，如图 3.2.19 所示。图中的迁移标签 start 表示单击 start 按钮这个事件，而 LAP 则表示单击 LAP 按钮。

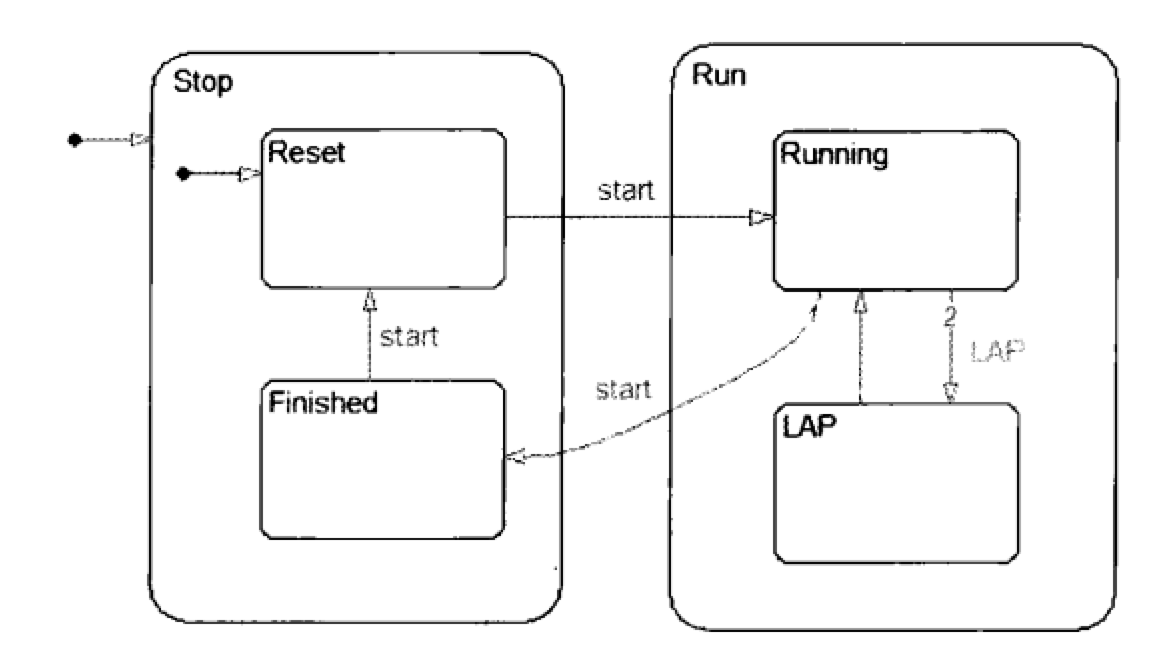


图 3.2.19　添加迁移及迁移标签

3.2.4　数据与事件

以圈计时需要 2 组数码管显示当前以及记录的分、秒、百分秒，另有 2 个按钮，为此需要添加 6 个数据与 2 个事件。数据是向外输出的，而事件是自外输入的。

添加数据或事件的方法有两种：使用菜单项 Add 或使用模型浏览器（Model Explorer）。前者的优势是添加方便，但菜单项仅提供了添加功能，无法通过菜单删除已添加的数据或事件，因此本书推荐用户使用模型浏览器。

1. 菜单项

在 Stateflow 编辑器窗口，选择菜单项 Add→Data→Output to Simulink，如图 3.2.20 所示。

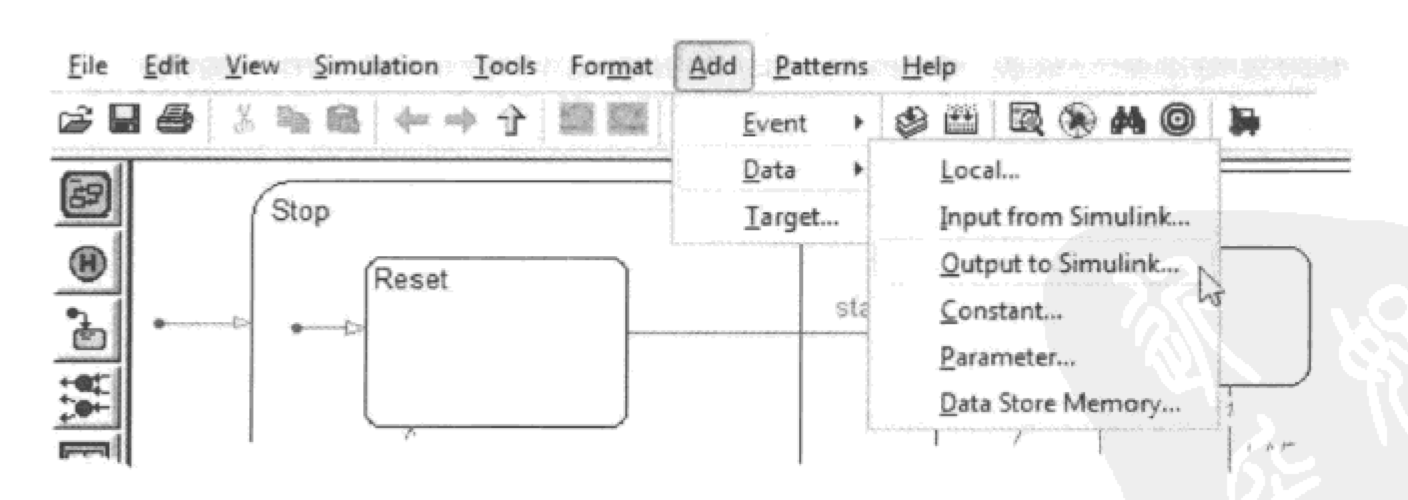


图 3.2.20　添加输出数据

在 Name 栏输入输出变量名 min，另外用户在 scope 栏还可以再次决定变量的作用域，如图 3.2.21 所示。

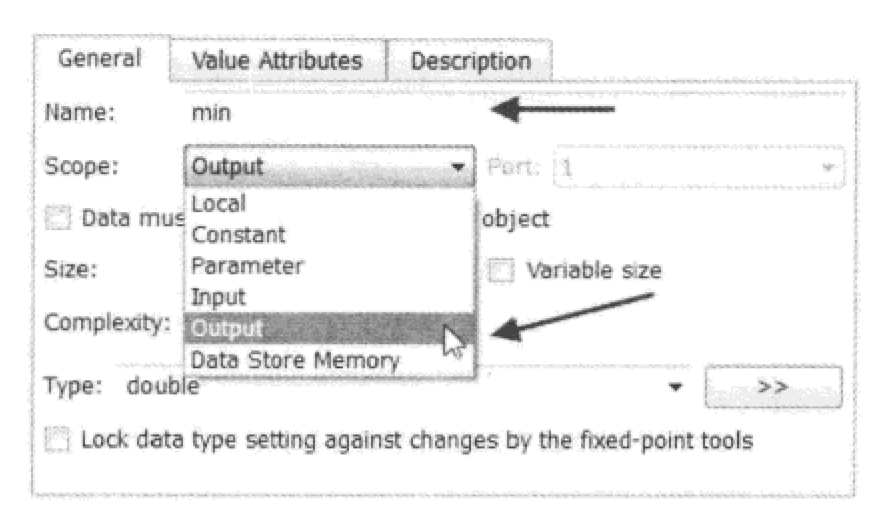


图 3.2.21　修改数据名及作用范围

2. 模型浏览器

在 Stateflow 状态图的顶层（即不选中任何图形对象），选择菜单项 Tools→Explore，或直接单击 Stateflow 编辑器窗口的工具栏按钮，打开模型浏览器，并确认已选中左侧模型结构图中的 Chart 节点，如图 3.2.22 所示。

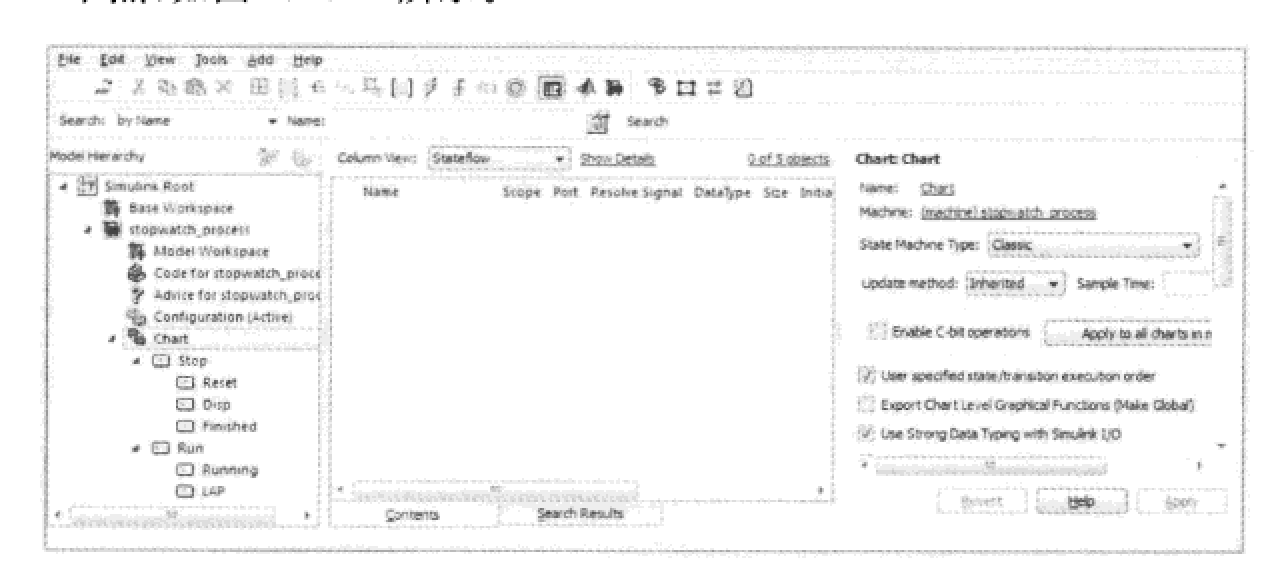


图 3.2.22　模型浏览器

在浏览器窗口的工具栏找到按钮、与✕，添加一个数据/事件或删除对应项。在中部窗口选中数据/事件的条目，右侧窗口即显示它的属性，如图 3.2.23 所示。与菜单项方法不同的是，使用浏览器添加的数据/事件，默认的作用域是本地（Local），用户需要手动修改为外部输入或外部输出。

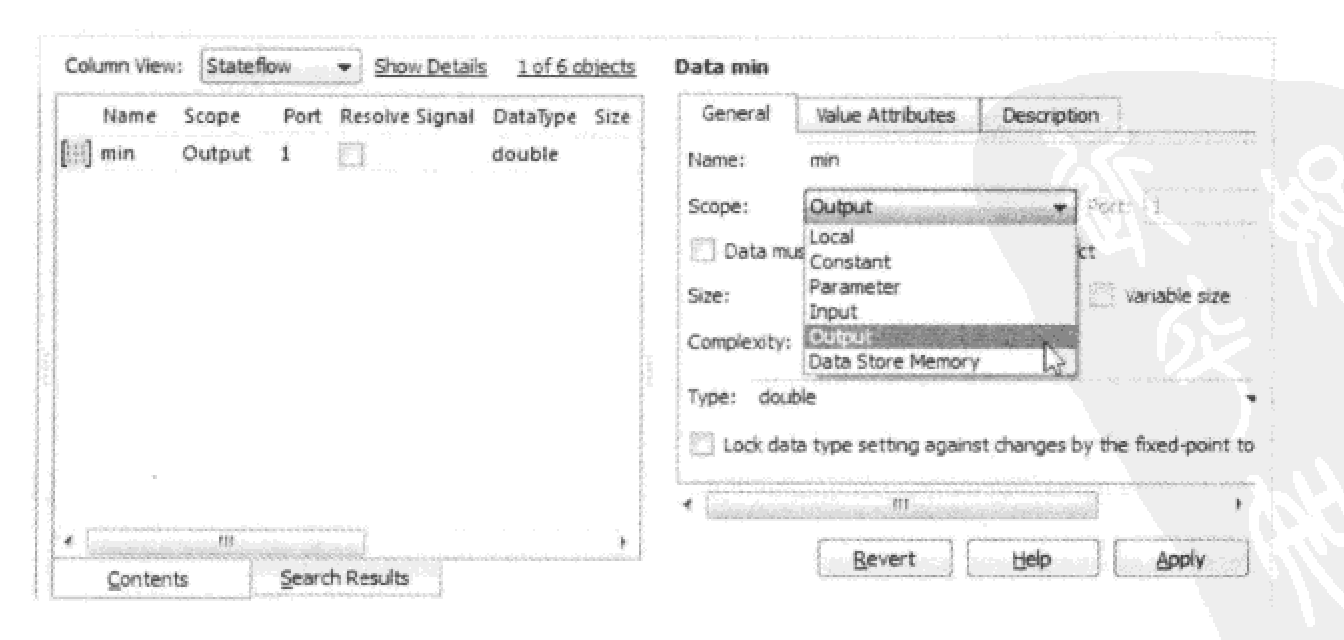
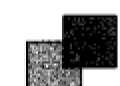


图 3.2.23　利用模型浏览器修改数据



对于事件，用户还需指定它的触发方式，Rising 表示上升沿、Failing 表示下降沿，而 Either 表示上升或下降沿皆可触发，本例的两个输入事件 start 与 LAP 皆选用 Either 方式触发，如图 3.2.24 所示。

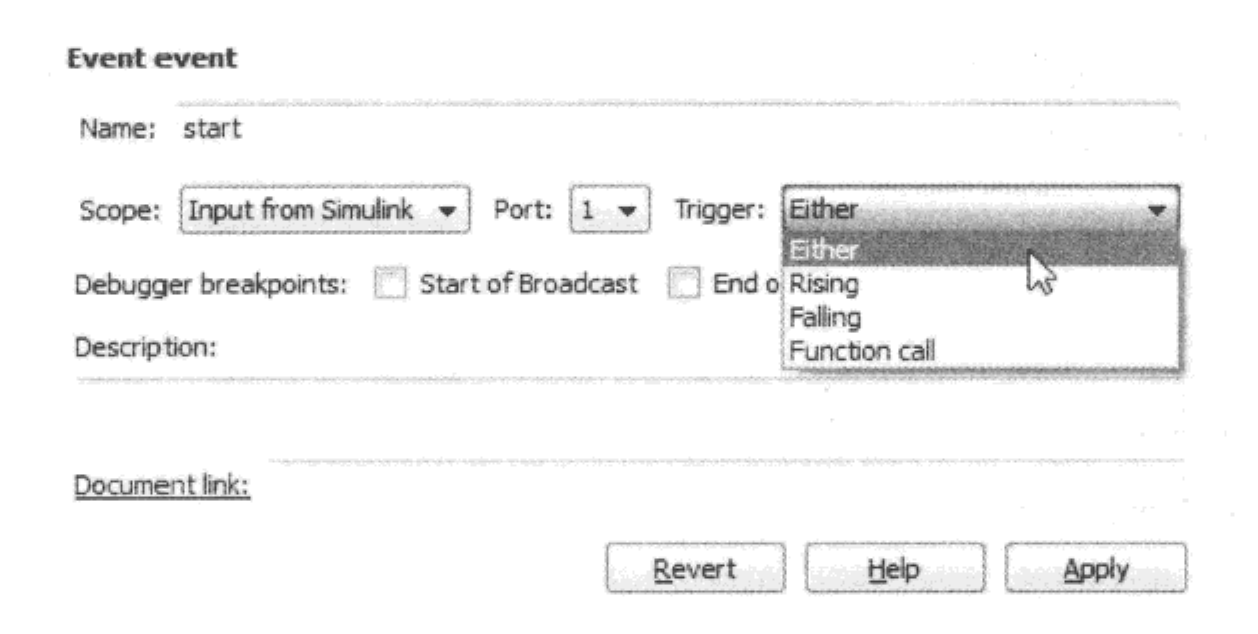


图 3.2.24　利用模型浏览器修改事件

有多个数据或事件时，用户还可以指定它们的端口号，合理地排列这些端口，将有利于以后的 Simulink 模块连线。图 3.2.25 所示是完整的数据与事件列表。

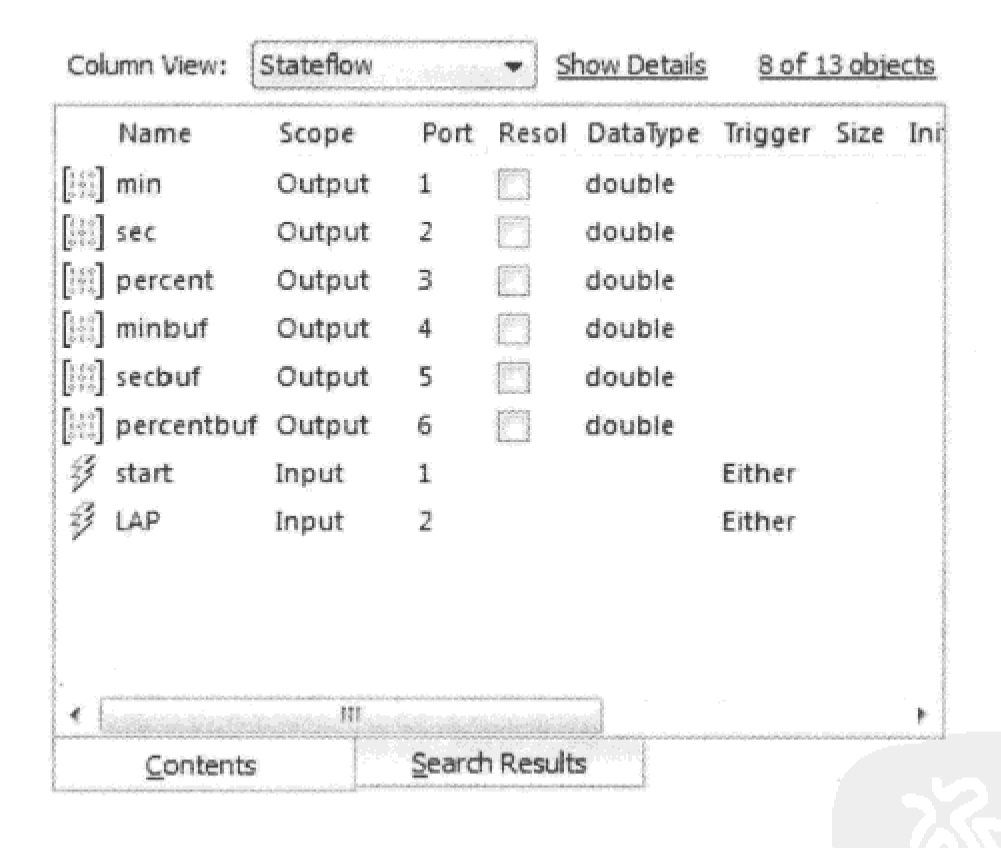


图 3.2.25　数据与事件列表

3.2.5　动　作

上文提到，显示时间值可以定义为状态动作，也可以定义为迁移动作。为不失一般性，本小节分别说明这两种动作的定义方法。

计时器复位时，两组数码管皆应清零，因此设置子状态 Reset 的进入动作代码如下：

```
Reset
entry:
min = 0;sec = 0;percent = 0;
minbuf = 0;secbuf = 0;percentbuf = 0;
```

单击 LAP 按钮或再次单击 start 按钮时，数码管 2 都必须显示当前时刻，因此设置子状态 LAP 与 Finished 的进入动作代码如下：

```
Finished
entry:
minbuf  =  min;
secbuf  =  sec;
percentbuf  =  percent;
```

添加了动作的状态图如图 3.2.26 所示。

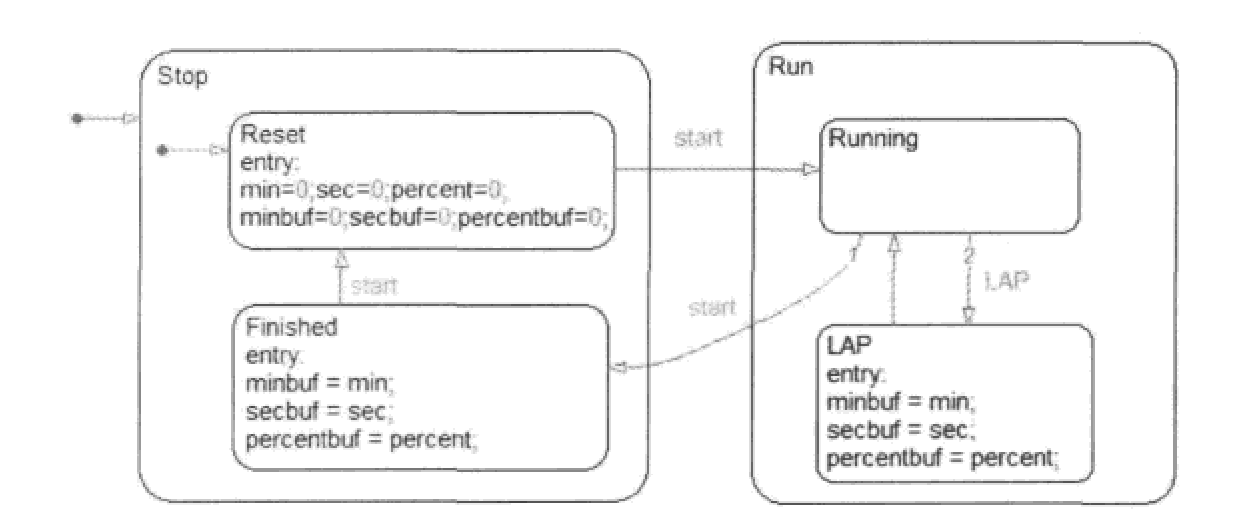


图 3.2.26　动作定义在状态

当然用户可以将显示时间值定义为迁移动作。为此按图 3.2.27 调整状态图，添加节点与迁移动作。对照图 3.2.26，用户应很容易理解图 3.2.27 的意义。

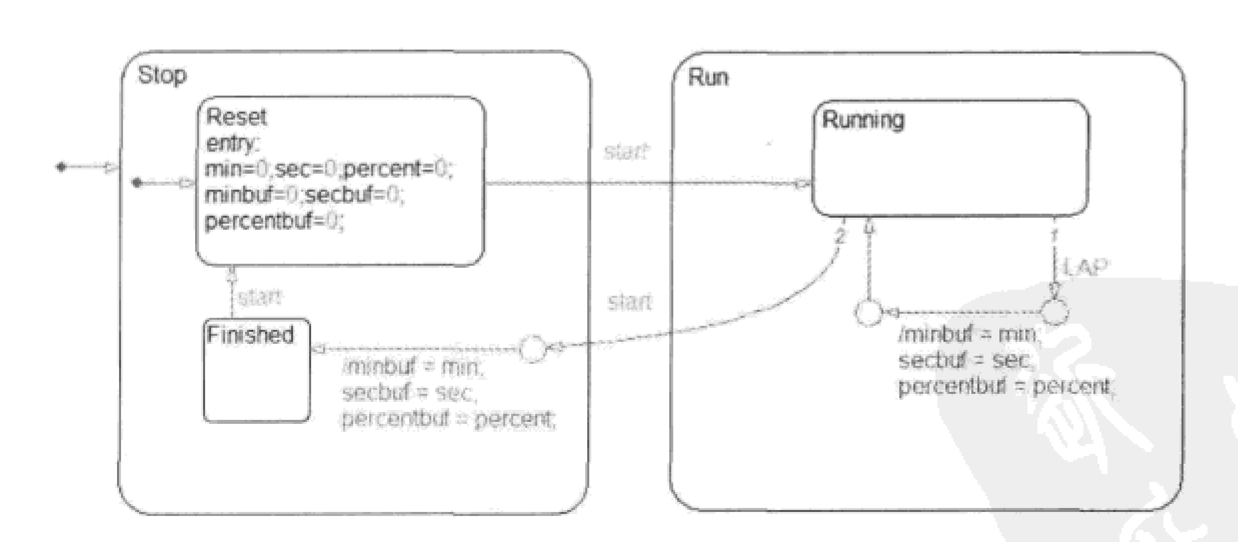
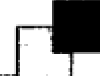


图 3.2.27　动作定义在迁移

不过目前的 Stateflow 状态图还缺少最关键的时钟程序，该过程详见 3.3 节。



3.2.6　自动创建对象

实际建模时，随着思路的不断扩展与成熟，状态、迁移、数据、事件、动作等各种图形与非图形对象，总是交替着添加，很少按照上述过程逐步进行。这就难免发生遗漏，尤其是大型、多层次的 Stateflow 状态图。

当用户完成状态图的编辑，单击“仿真”按钮时，系统首先要进行语法检查，如果发现错误，则给出提示。如果是某些数据或事件未定义，用户可使用随后出现的 Symbol Autocreation Wizard 向导，自动创建缺失的对象。

例如，用户可以在图 3.2.26 中任意添加一个状态动作 $y=1$，任意添加一个迁移事件 stop，如图 3.2.28 所示。

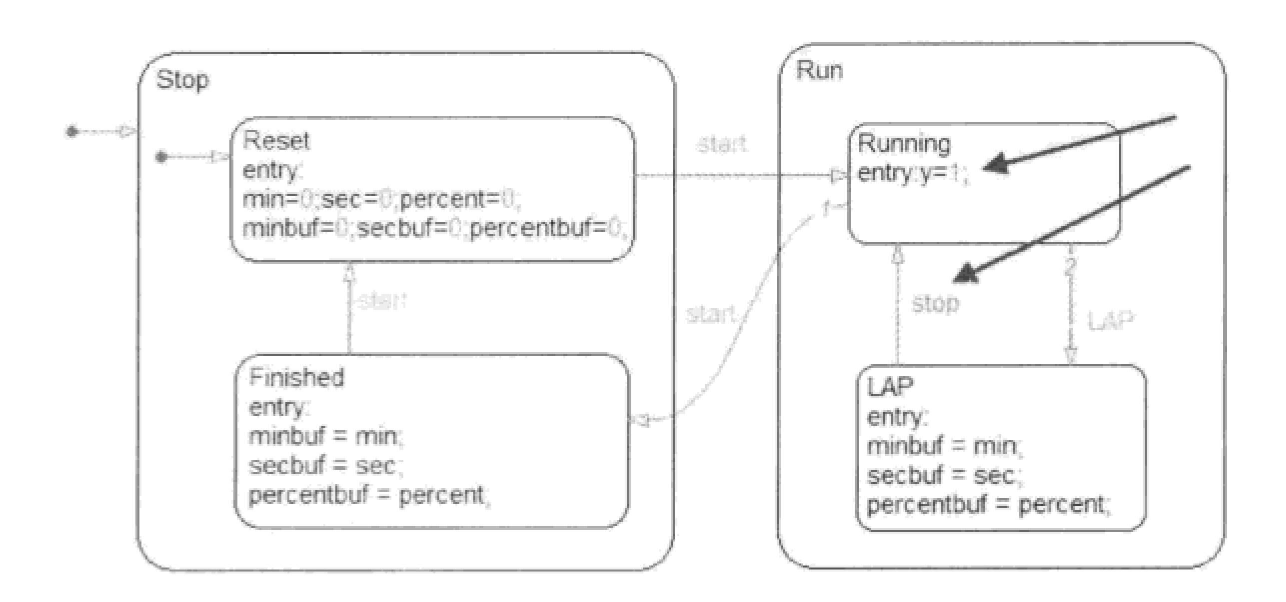


图 3.2.28　添加一个状态动作

单击“仿真”按钮，系统首先给出语法错误提示，如图 3.2.29 所示。

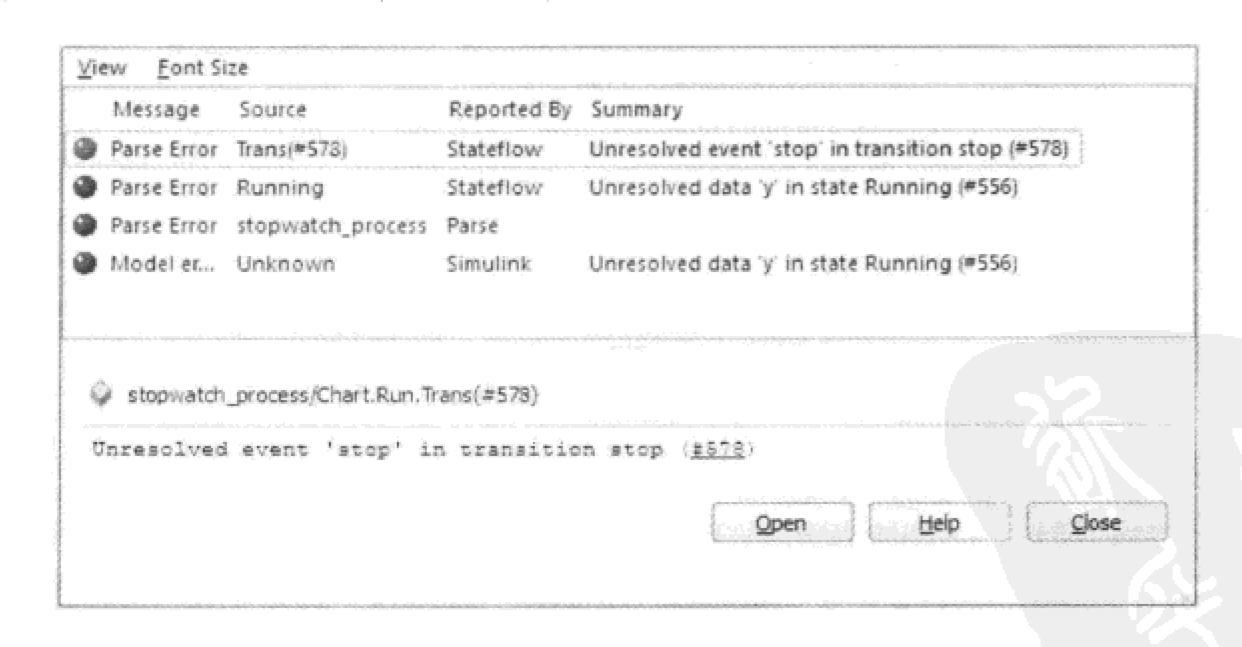


图 3.2.29　错误提示

系统进一步分析该语法错误，是由于状态图中存在无法处理的符号，因此弹出 Symbol Autocreation Wizard 向导与向导使用说明，如图 3.2.30 所示，若用户已能够熟练使用，可忽略该说明。

Symbol Autocreation Wizard 向导建议应另行创建数据 y 与事件 stop，如图 3.2.31 所示。

用户可以不断单击 Scpoe 与 Proposed Parent 的内容，修改数据/事件的作用域与父对象，如图 3.2.32 所示。

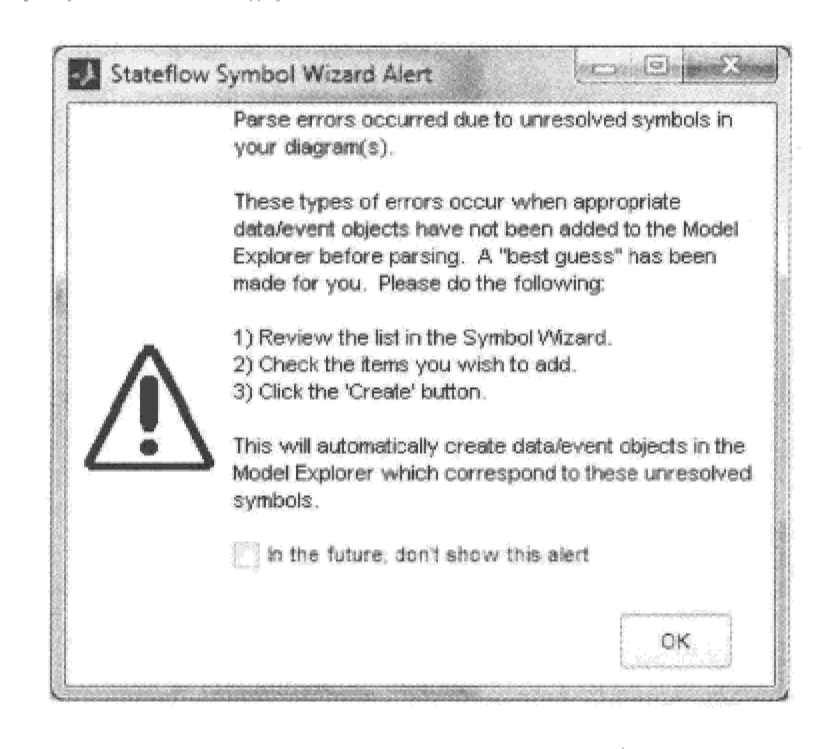


图 3.2.30　向导使用说明



图 3.2.31　自动创建对象向导

图 3.2.32　修改作用域与父对象

用户若接受向导的建议，则单击数据/事件前的空白处，选中该条目，之后单击 Create 按钮，创建对象，如图 3.2.33 所示。

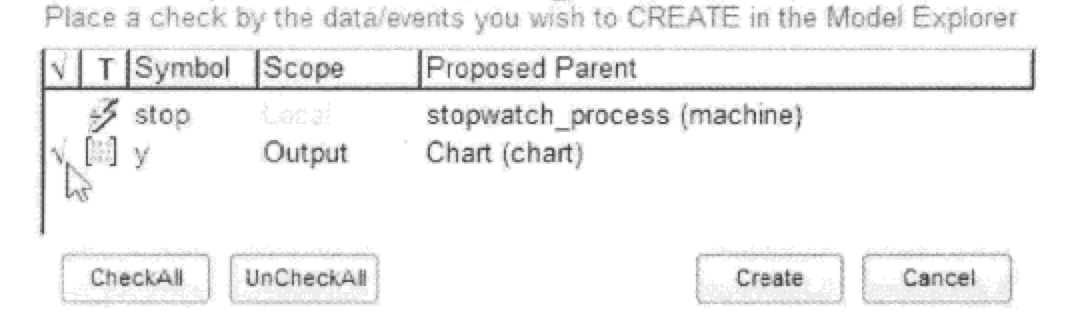


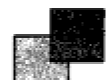
图 3.2.33　创建对象

3.3　Stateflow 流程图

3.3.1　流程图与节点

3.1 节与 3.2 节介绍了 Stateflow 状态图的基本概念与创建过程。状态图的一个特点是，在进入下一个仿真步长前，它会记录当前的本地数据与各状态的激活情况，供下一步长使用。而流程图只是一种使用节点与迁移来表示条件、循环、多路选择等逻辑的图形，它不包含任何的状态。

由于迁移（除了默认迁移）总是从一个状态到另一个状态，节点之间的迁移只能是一个迁移段。因此流程图可以看作是有若干个中间支路的一个迁移，一旦开始执行，就必须执行到终节点（没有任何输出迁移的节点），不能停留在某个中间节点，也就是说必须完成一次完整的



迁移。

从另一个角度来看，节点可以认为是系统的一个判决点或汇合点，它将一个完整的迁移分成了若干个迁移段。因此可以将几个相同的迁移段合并为一个，用一个迁移表示多个可能发生的迁移，简化状态图，由此生成的代码也更加有效。

对于以下情况，用户应首先考虑使用节点：

(1) if-else 判断结构、自循环结构、for 循环结构。

(2) 单源状态到多目标状态的迁移。

(3) 多源状态到单目标状态的迁移。

(4) 基于同一事件的迁移。

注意：事件无法触发从节点到状态的迁移。

建议：用户可以把流程图封装成一个图形函数（详见 3.6.2 节），便于在 Stateflow 的任意位置调用。

3.3.2　建立流程图

1. 手动建立

建立流程图的过程与建立状态图的过程相似，以一段简单的代码为例，手动建立流程图。

```
if percent == 100
    {percent = 0;
    sec = sec + 1;}
else if sec == 60
        {sec = 0;
        min = min + 1;}
    end
end
```

(1) 起始节点。

单击按钮，添加起始节点。如图 3.3.1 所示。

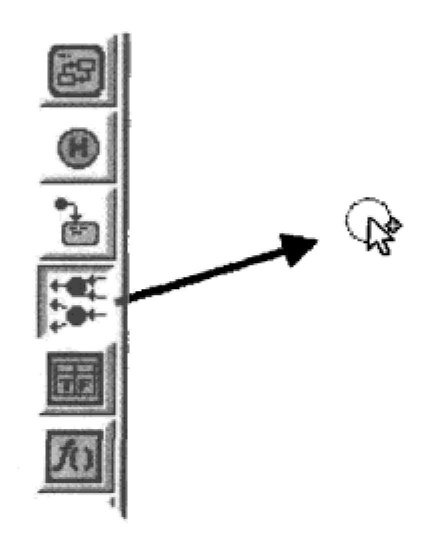


图 3.3.1　添加节点

(2) 条件节点与终节点。根据代码的执行过程，逐一添加条件节点 A1、B1、C1，终节点 A2、B2，以及节点间的迁移与迁移标签，如图 3.3.2 所示。

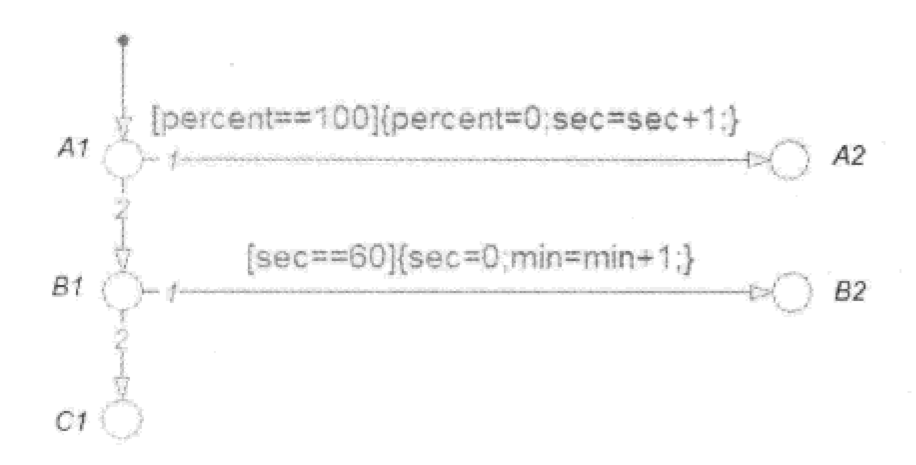


图 3.3.2 流程图

流程图运行过程如下：

① 系统默认迁移进入节点 A1，如果条件[percent == 100]为真，执行{percent＝0；sec＝sec＋1；}，并向终节点 A2 迁移。

② 如果条件[percent == 100]不为真，向 B1 节点迁移，继续判断如果条件[sec == 60]为真，执行{sec＝0；min＝min＋1；}，并向终节点 B2 迁移；

③ 如果不满足任何条件，则向终节点 C1 迁移。

(3) 节点与箭头大小。对于某些重要的节点或迁移，用户可以调整其节点大小与迁移箭头的大小，突出其地位。例如，选择节点 C1 的右键菜单项 Junction Size→16，如图 3.3.3 所示，放大节点；选择节点 A1 的右键菜单项 Arrowhead Size → 20，放大指向该节点的所有迁移箭头，如图 3.3.4 所示。

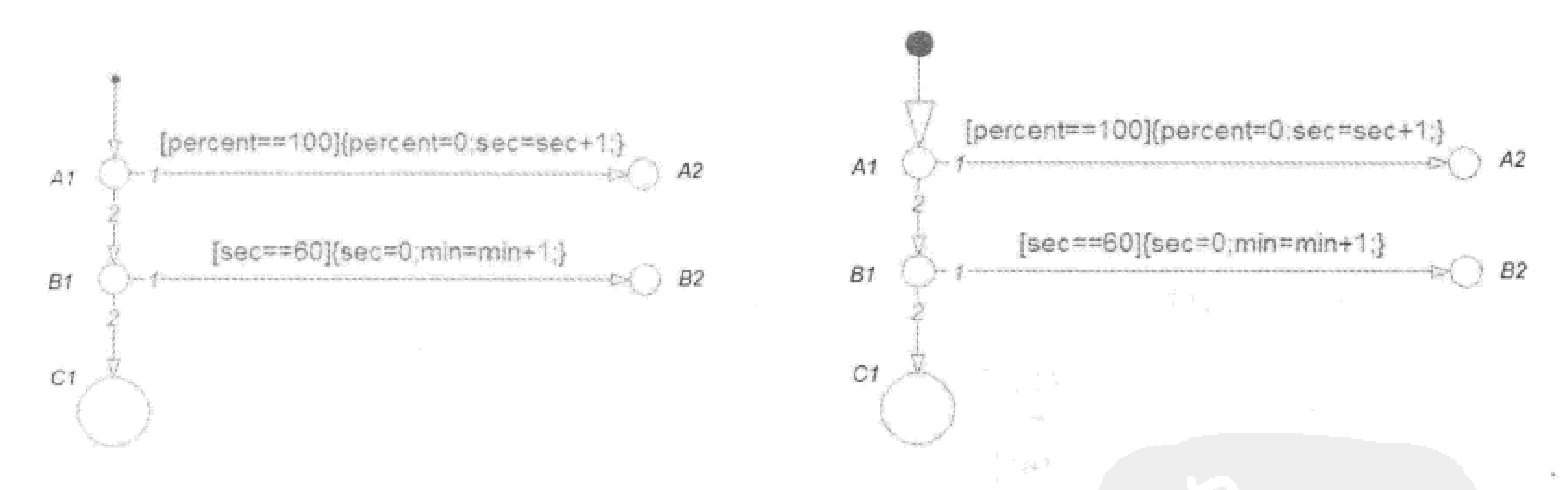
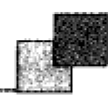


图 3.3.3 节点大小　　　　**图 3.3.4 箭头大小**

(4) 优先级。两个判断节点 A1、B1，均有两条输出迁移，分别标记了数字 1、2，这表示迁移的优先级。默认情况下，Stateflow 状态图使用显性优先级模式，用户可以自行修改各个迁移优先级。

例如，选择迁移曲线的右键菜单项 Execution Order，将优先级由 1 降低为 2，如图 3.3.5 所示。修改了某一输出迁移的优先级，系统会自动调整同一节点另一迁移的优先级。

为避免用户错误地设置优先级，Stateflow 提供了另一种模式：隐性优先级。选择编辑器菜单项 File → Chart Properties，取消勾选 User specified state/transition execution order 复



选框，启用隐性模式，如图 3.3.6 所示。

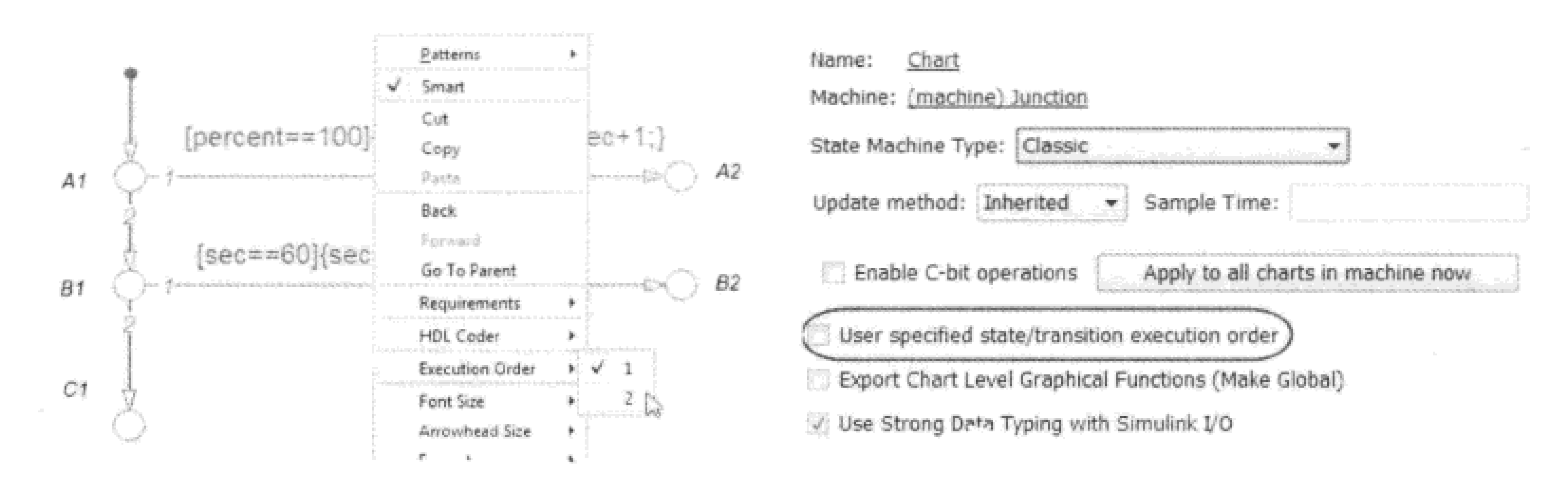


图 3.3.5　迁移优先级　　　　图 3.3.6　自动设置迁移优先级

使用这种模式时，系统根据以下规则，自动设置迁移优先级，从高到低排列如下：

① 既有事件又有条件的迁移。

② 仅有事件的迁移。

③ 仅有条件的迁移。

④ 不含任何限制的迁移。

注意：同一个 Stateflow 状态图，只能选用一种优先级模式，但对于有多个状态图的 Simulink 模型，则不受此限制。

2. 自动建立

对于简单的流程图，手动建立难度不大，而对于稍复杂的逻辑，用户难免会感到无从下手。Stateflow 提供了快速建立流程图的向导，它可以生成三类基本逻辑：判断、循环、多条件。本小节使用向导，重建图 3.3.2 的流程图。

(1) 单击编辑器菜单项 Patterns→Add ...，选择流程图的类型，如图 3.3.7 所示。

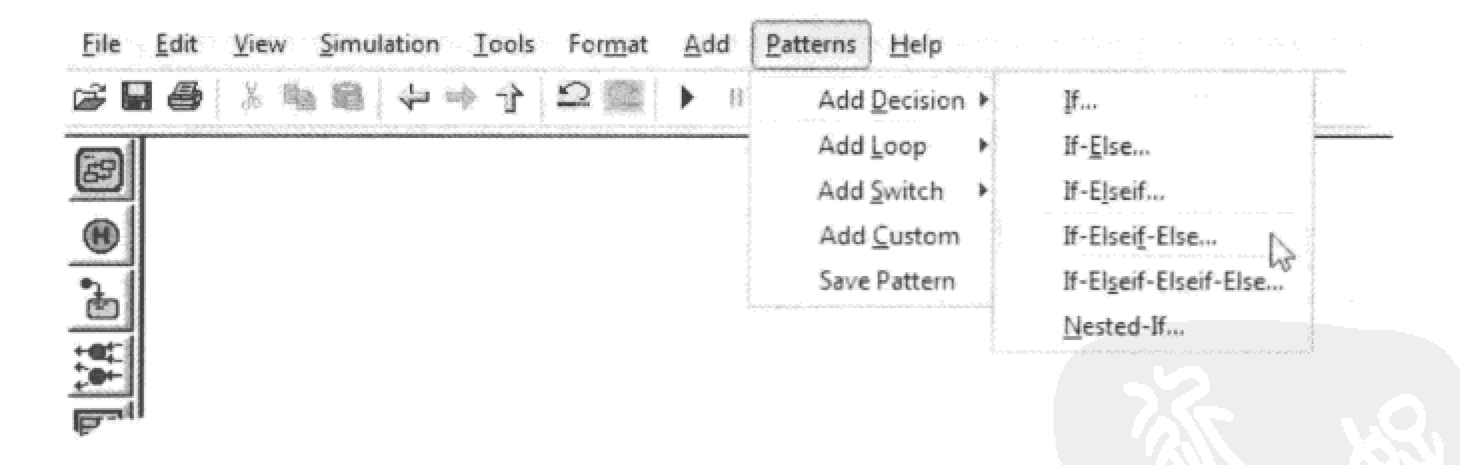


图 3.3.7　流程图向导菜单

(2) 这里选择 Patterns → Add Decision → If-Elseif-Else...，在随后打开的对话框中输入判断条件与对应的动作，如图 3.3.8 所示。

(3) 生成的流程图如图 3.3.9 所示。

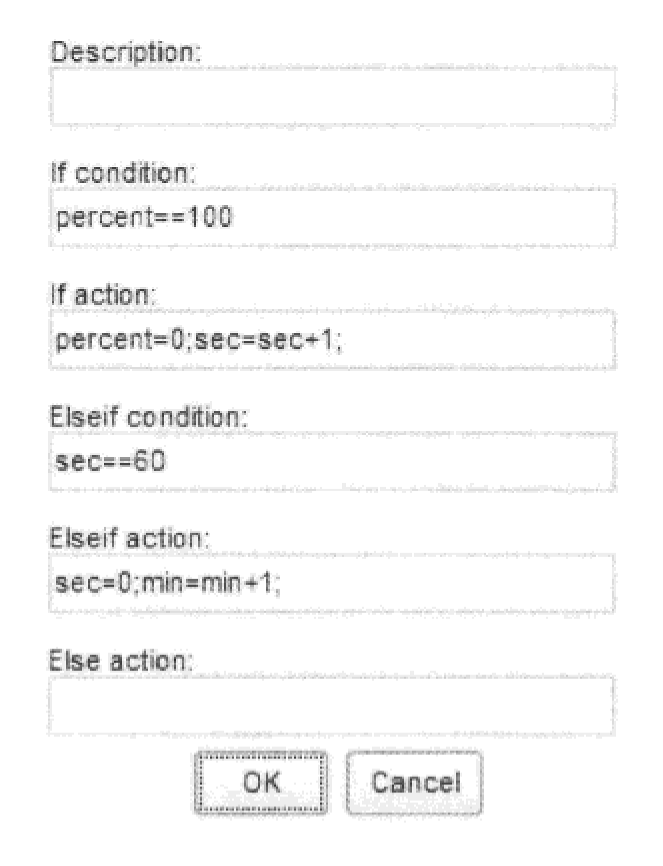


图 3.3.8 新建流程图对话框

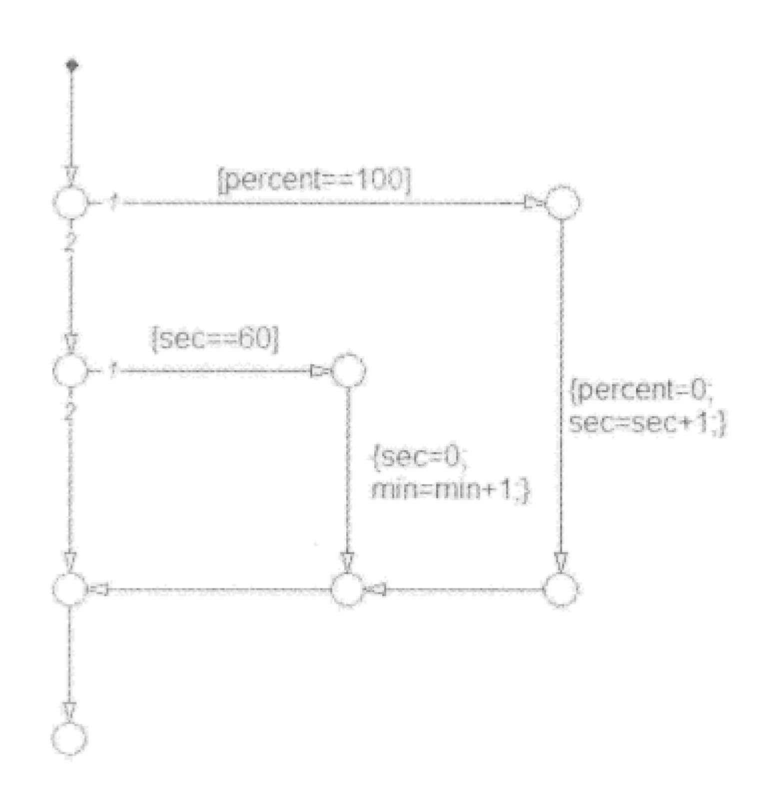


图 3.3.9 流程图

3. 两种方式的对比

尽管用户可以手动建立流程图，但使用流程图向导的优势也是显而易见的：

(1) 任何一种流程图都可归结为判断、循环、多条件，或者三者的组合，因此皆可以使用向导自动生成。

(2) 使用向导生成的流程图符合 MAAB(MathWorks Automotive Advisory Board)规则，这有利于后期模型检查。

(3) 各种流程图的外观基本一致。

(4) 将设计好的流程图，另存为模板，便于重用。

3.4 层次结构

3.4.1 层次的概念

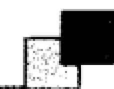
Stateflow 的对象具有层次性，一个 Stateflow 对象可以包含其他 Stateflow 对象，例如状态内若包含其他状态，则形成父状态，其内部状态称为子状态。当状态具有第二个层次时，状态就构成了层次。

状态具有了层次，迁移自然也具有了层次，Stateflow 允许在不同层次状态之间存在转移。如果迁移穿越了父状态的边界直接到达了低层次的子状态，称为超迁移。

在状态图中使用层次有以下几个目的：

(1) 使用层次，可以将相关的对象组合在一起，构成族群。

(2) 可以将一些通用的迁移路径或者动作组合成为一个迁移动作或路径，简化模型。



(3) 适当地使用层次,可以有效地缩减生成代码的大小,也能够提高程序执行的效率和可读性。

3.4.2 迁移的层次

1. 内部迁移

内部迁移是指从父状态边缘内部出发,终止于子状态外边缘的迁移,迁移始终处于父状态的内部,不会退出源状态。

在交通灯系统中,同一个父状态 PowerOn 存在红黄绿三个子状态,它们需要不停地转换,但除非发生 PowerOff 事件,不会退出父状态,对于这样的逻辑过程,读者可能习惯使用节点将三种状态的迁移联系起来,如图 3.4.1 所示。

使用了内部迁移,可直接从父状态激活相应的子状态,不必经过节点,大大简化状态图,如图 3.4.2 所示。

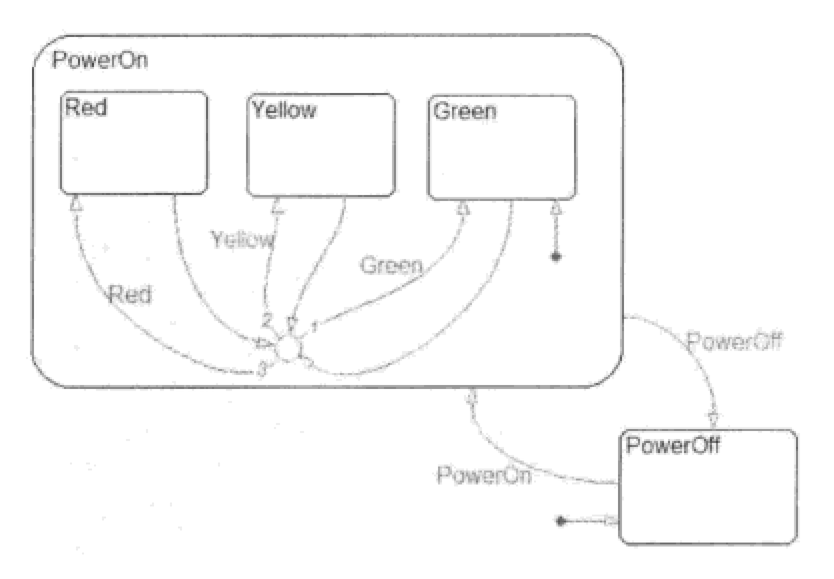


图 3.4.1 带有节点的迁移

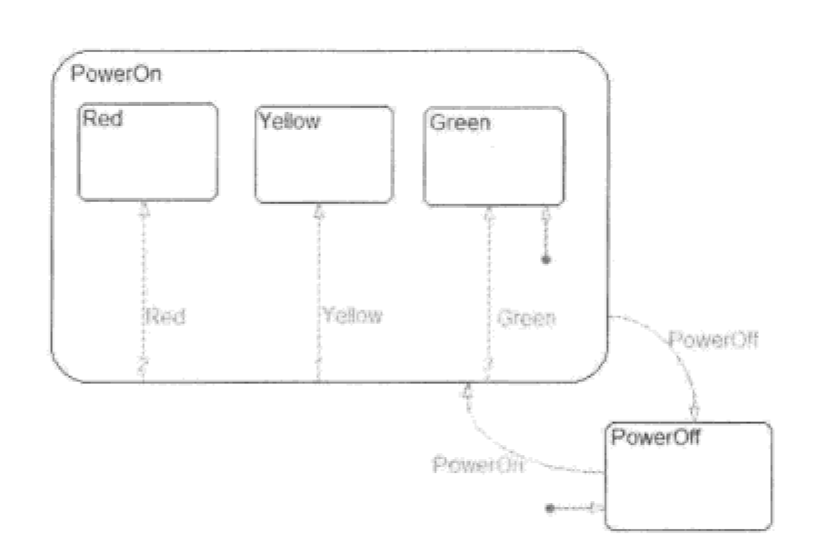


图 3.4.2 内部迁移状态图

2. 层次化迁移的优先级

与状态类似,迁移也具有层次性,迁移所属的层次是由其父状态、源状态和目标状态决定的。因此,当多个迁移同时有效时,Stateflow 需要有一个层次化迁移优先级机制来判断迁移顺序。

层次化迁移的优先级规则为:从高层次到低层次检测;从外部迁移到内部迁移检测;同一层次内,超转移优先。

如图 3.4.3 所示,图表激活时,默认迁移激活状态 A,继而状态 A.a1 被次级默认迁移激活,这时按以下优先级检测迁移是否有效:

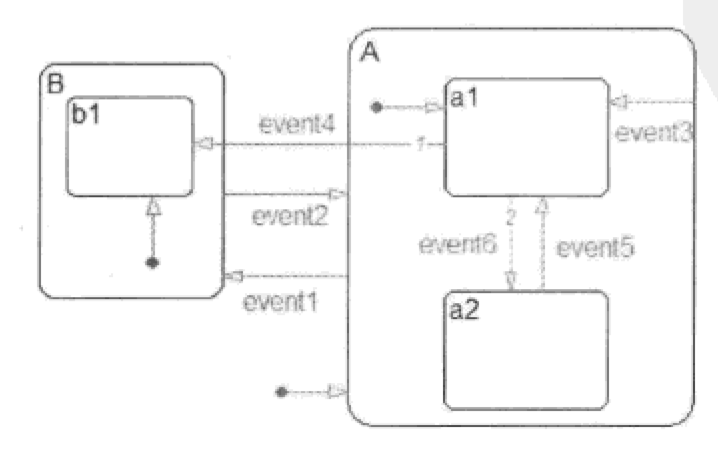


图 3.4.3 层次化的迁移

(1) 检测高层 A,B 状态的外部迁移是否有效(event1,event2)。

(2) 检测高层 A,B 状态的内部迁移是否有效(event3)。

(3) 检测低层 a1,a2,b1 状态的超迁移是否有效(event4)。

(4) 检测父状态 A 内部子状态间的迁移是否有效(event5,event6)。

3.4.3 历史节点

在状态图的顶层或一个父状态里放置一个历史节点,它便能记录退出父状态时,正处于激活状态的子状态,当再次进入父状态时,则默认激活上一次所记录的子状态,而不是激活默认迁移的状态。

历史节点的作用域仅限于它所存在的层级。

如图 3.4.4 所示,父状态 A2 中加入了历史节点,因此当第 1 次激活 A2 状态时子状态 C1 被激活,满足迁移条件时 C2 被激活,但此后 A2 状态向 A1 状态的迁移将优先发生,C2 状态不再向 C1 状态迁移。于是第 1 次激活 A2 状态时,被激活的子状态是 C2,而不是 C1。读者可以从图 3.4.5 所示的输出看出上述的迁移过程。

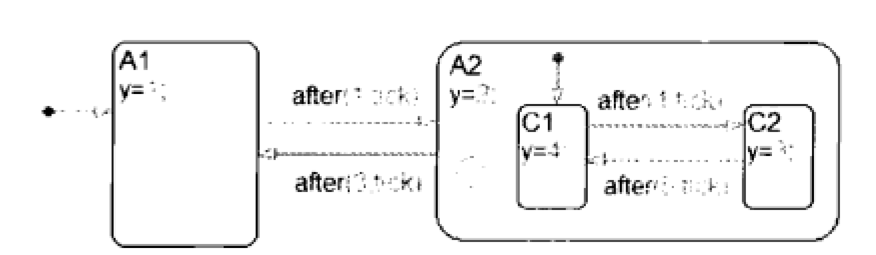


图 3.4.4 历史节点

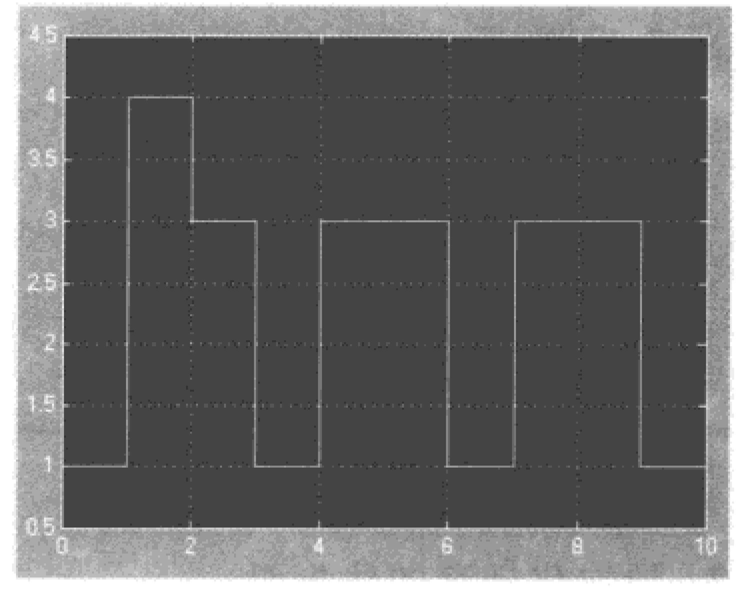


图 3.4.5 运行结果

图 3.4.6 与图 3.4.7 是不包含历史节点的状态图及其运行结果,用户应详细比较两者的区别。

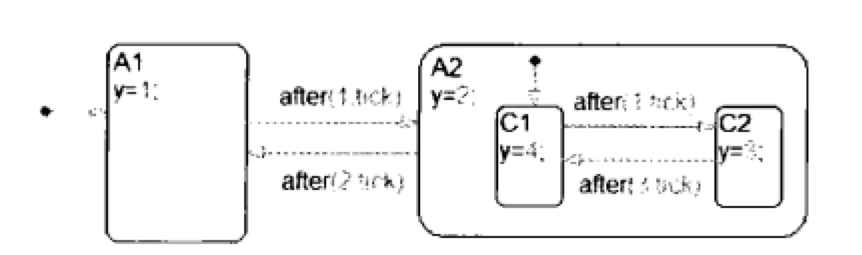


图 3.4.6 不含历史节点的状态图

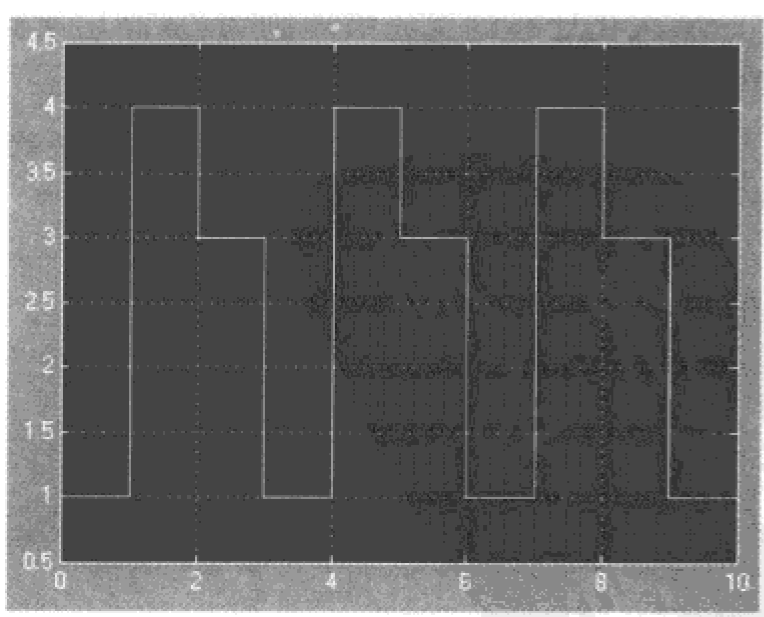


图 3.4.7 运行结果



3.4.4 子状态图

子状态图其实就是其内部所包含的状态图的父状态。用户可以像操作父状态一样为子状态图定义状态动作,迁移和默认迁移。与 Simulink 中的子系统类似,子状态图隐藏了模型的细节,简化了图表。

以图 3.4.8 为例,在状态 A 中右击,在右键菜单中选择 Make Contents→Subcharted 命令。此时,A 状态变成了灰色,隐藏了内部细节,成为子状态,如图 3.4.9 所示。用户再次进行 Make Contents→Subcharted 操作,则可取消子状态图。

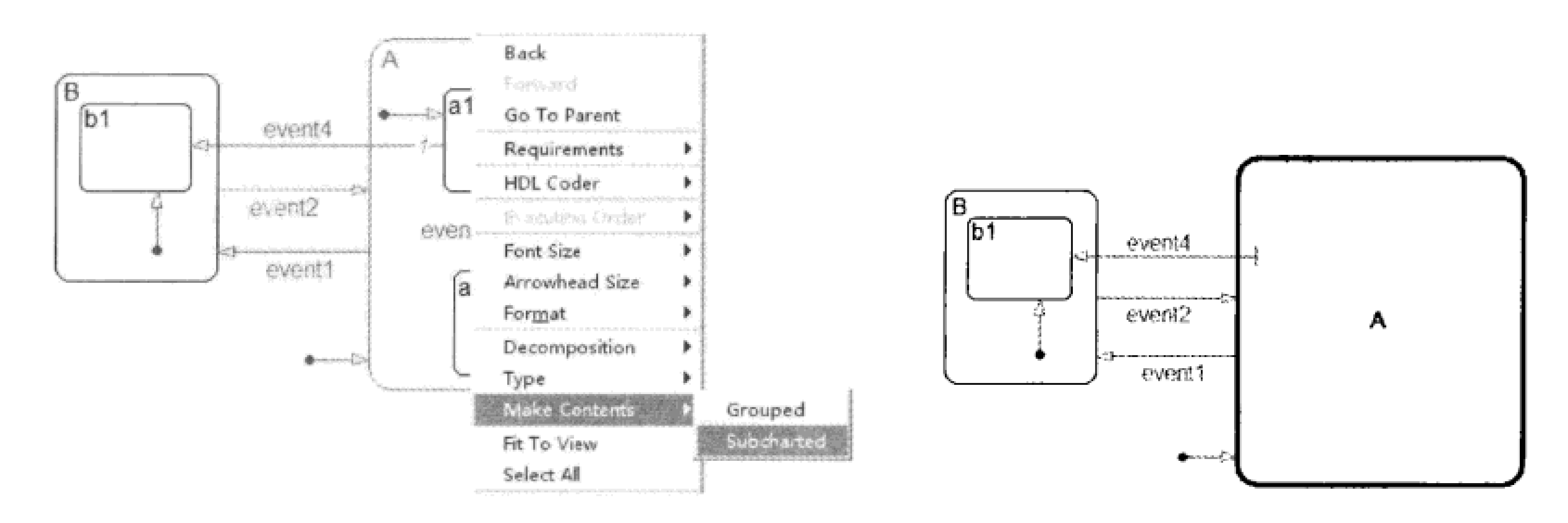


图 3.4.8 建立子状态图　　　　图 3.4.9 子状态图

双击子状态图 A 可以看到其内部细节。利用 Stateflow 编辑器上的三个箭头工具按钮,用户可以方便地在各层次图表间进行切换,如图 3.4.10 所示。

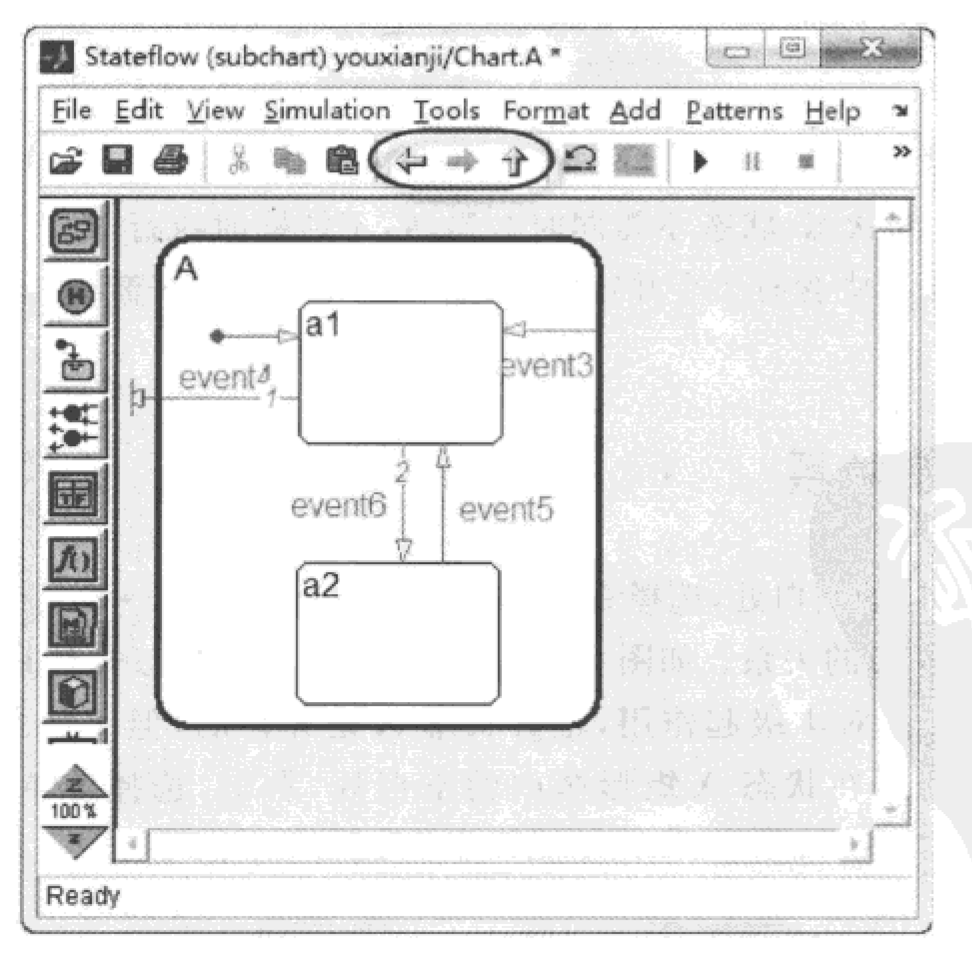


图 3.4.10 子状态图内部细节

3.4.5 层次状态图中的流程图

流程图中不包含任何状态，也不保留任何状态信息。流程图一旦激活，即由默认迁移一直运行到终止节点为止。

如图 3.4.11 所示，当状态 A 处于激活状态时，B 事件发生则激活状态 B，这时便执行状态 B 中的流程图，并且只执行一次，执行完成后 B 状态仍保持激活，直到发生事件 A，退出 B 状态。

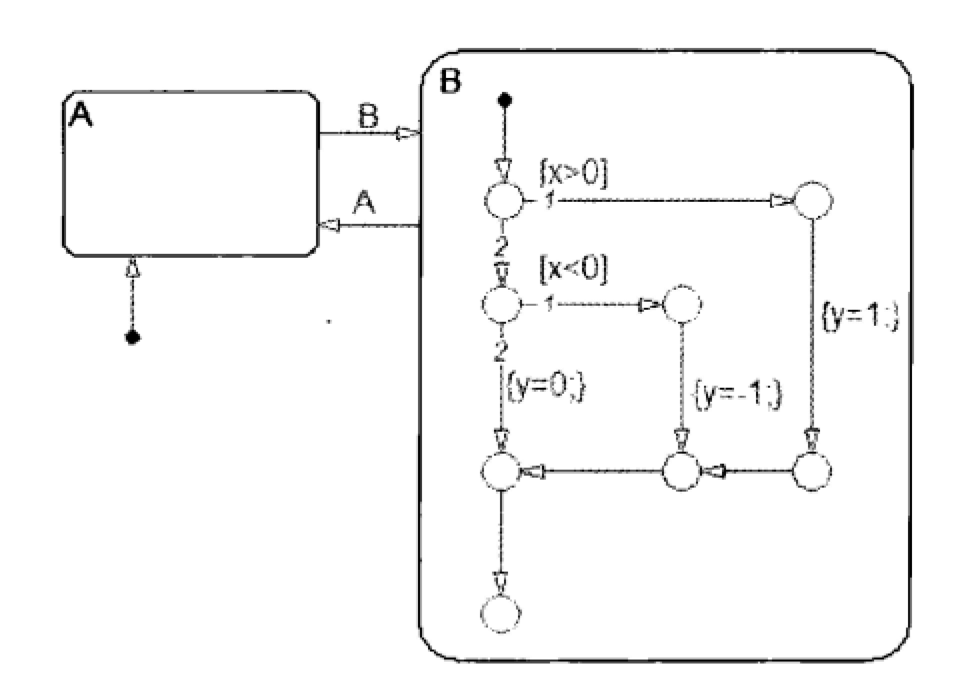


图 3.4.11 状态中的流程图

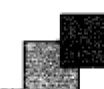
3.5 并行机制

状态可分为两大类：互斥状态 exclusive (OR)和并行状态 parallel (AND)。若在同一个层次中含有多个互斥的状态，状态不能同时被激活，不能同时执行，在 Stateflow 中用实线框表示。相反，若同一层次中含有多个并行状态，则一旦父状态处于激活状态，其并行子状态同时处于激活状态。

3.5.1 设置状态关系

在状态图编辑窗口的空白处，选择右键菜单项 Decomposition→Exclusive(OR)或 Parallel (AND)，可设置顶级状态的关系。如图 3.5.1 所示，状态 A 与状态 B 是并行的。

状态关系的设置仅对本级起作用，尽管状态 A 是并行的，但子状态 A1、A2 仍是互斥的，若要修改，用户需要在状态 A 矩形框内的空白处，右击，选择菜单项 Decomposition→Parallel(AND)，如图 3.5.2 所示。



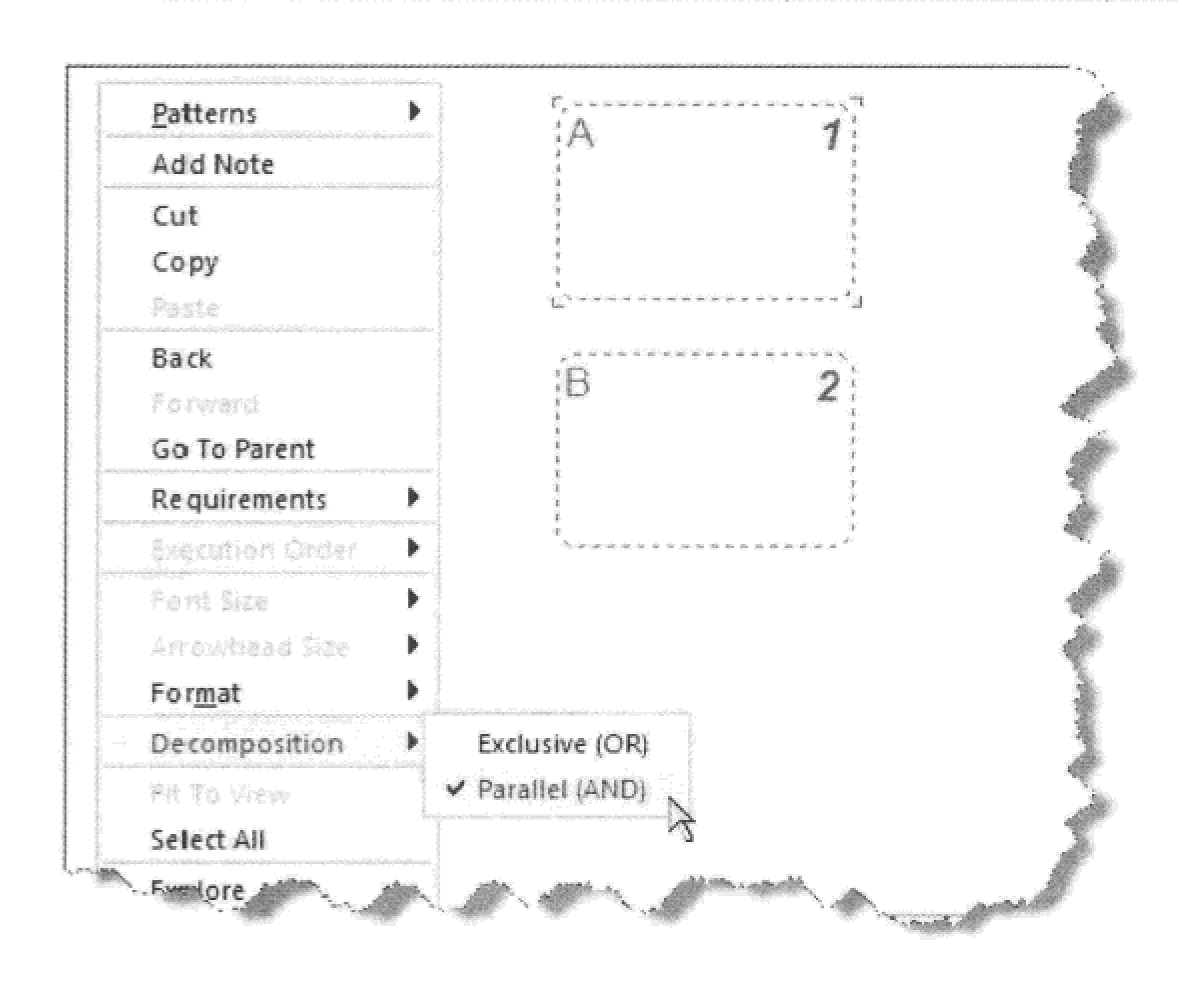


图 3.5.1 设置顶级状态关系

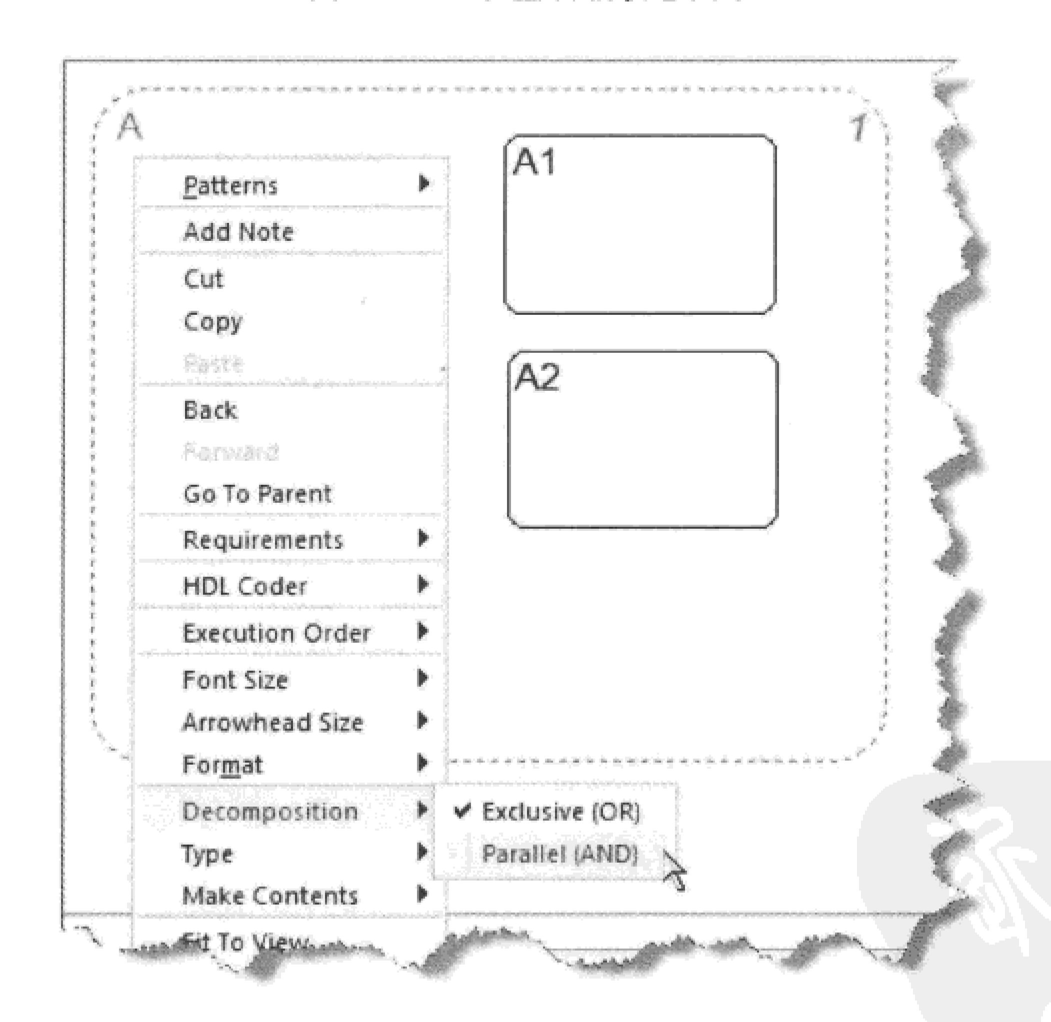


图 3.5.2 修改状态关系

3.5.2 并行状态活动顺序配置

处于同一层次下的所有并行的状态应该在其父状态被激活的时候同时被激活，但是它们

的激活是按照一定顺序进行的。其默认激活顺序为:从上到下,从左到右。并且在每个状态的右上角用数字标注,如图 3.5.3 所示。

若用户希望改变其激活顺序。例如将 LED 状态放在 Fan 之后激活,可以先选中 LED 状态,在右键菜单中的 Excution Order 子菜单中改变激活顺序,如图 3.5.4 所示。

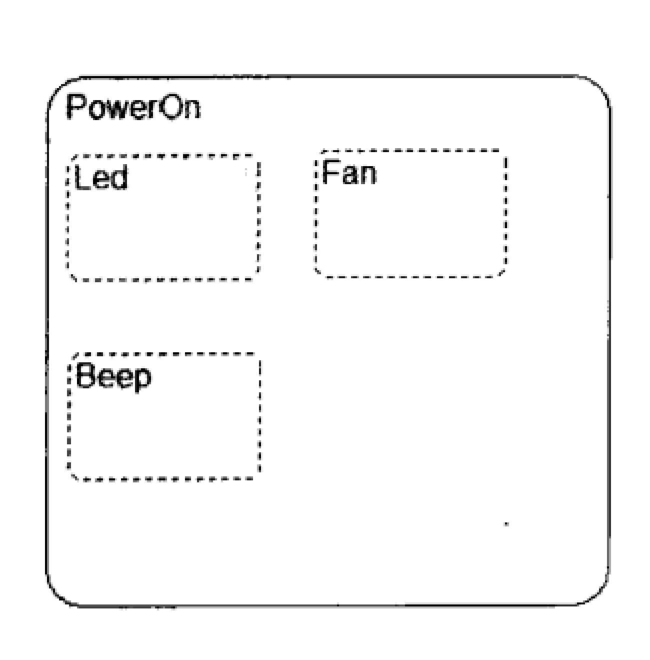


图 3.5.3 默认激活顺序

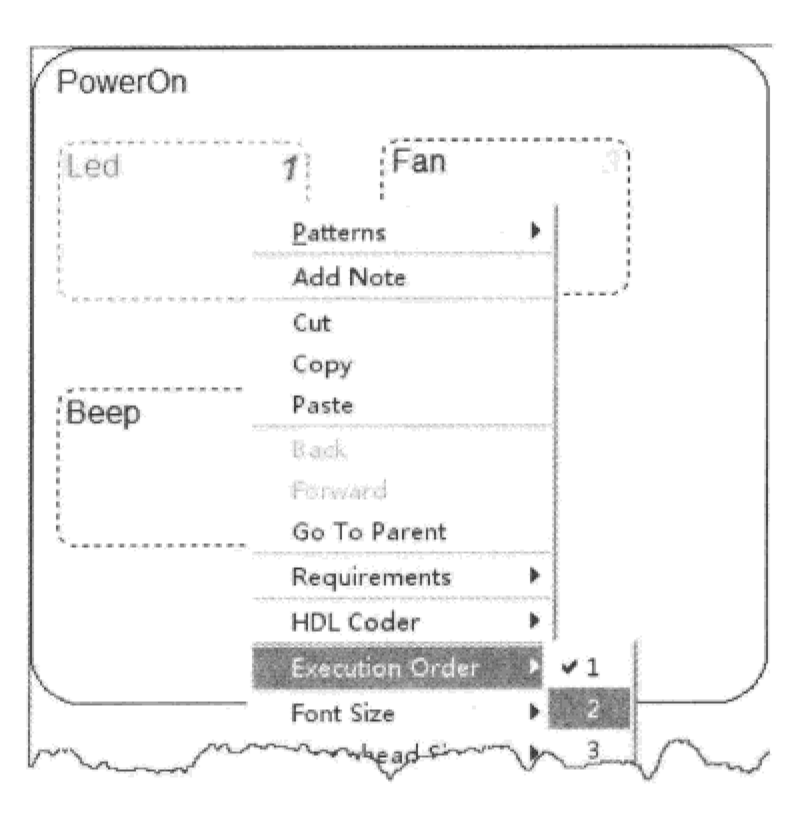


图 3.5.4 改变激活顺序

3.5.3 本地事件广播

使用本地事件广播,可以在某个状态内部触发其他并行状态的执行,这样就可以在系统的不同状态之间实现交互,让一个状态的改变影响其他状态。事件广播可以触发状态动作、迁移动作和条件动作。使用广播之前,需要预先定义事件。

如图 3.5.5 所示,Led 状态和 Fan 状态为并行关系。

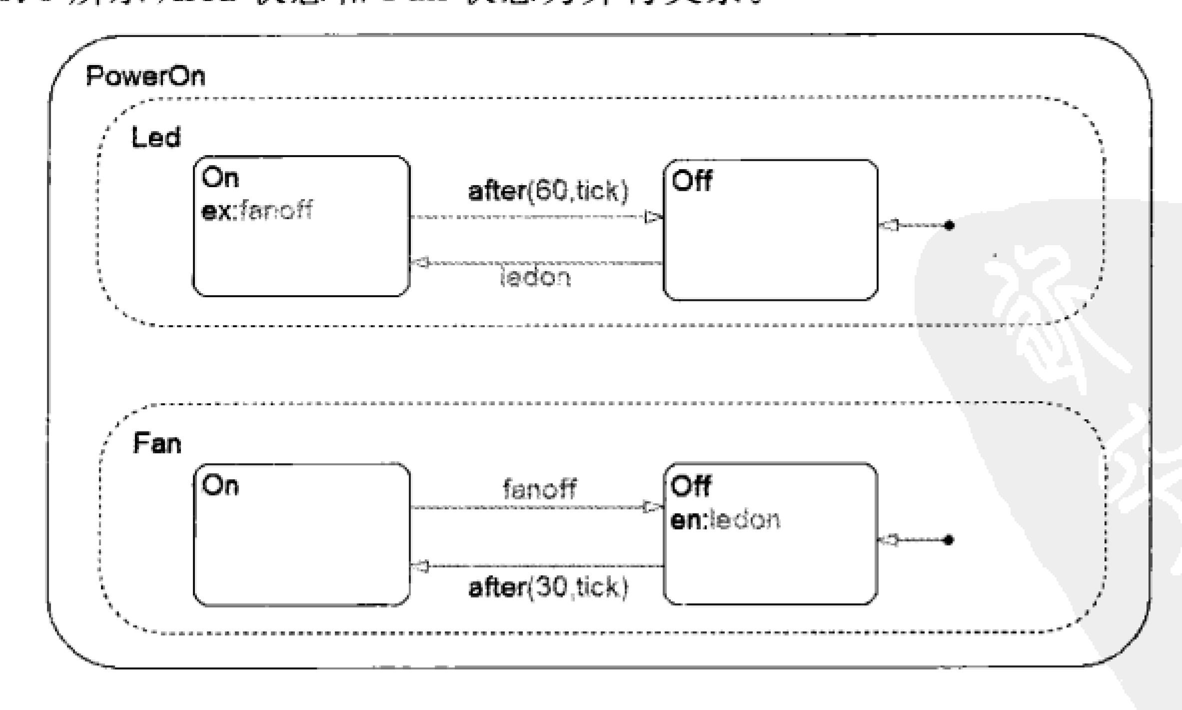
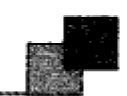


图 3.5.5 本地事件广播



(1) 当父状态PowerOn激活时，Led.Off状态和Fan.Off状态同时被激活，Fan.Off状态广播事件ledon，于是Led.Off状态向Led.On状态迁移。

(2) 当Led.On状态满足迁移条件退出时，广播事件fanoff，而此时Fan.Off状态已向Fan.On迁移，于是响应事件fanoff，向Fan.Off状态迁移。

3.5.4 直接事件广播

使用直接事件广播可以避免在仿真过程中出现不必要的循环或递归、并能有效地提高生成代码的效率。

1. 用send函数直接事件广播

send函数的完整格式为：send(event_name, state_name)，使用该函数进行直接事件广播，如图3.5.6所示。Stateflow执行过程如下：

(1) 当并行超状态Led和Fan激活时，对应的子状态Led.L1和Fan.F1被激活，子状态Led.L1执行状态动作flag=1。

(2) Led.L1至Led.L2的迁移条件[flag==1]为真，于是执行条件动作send(event_1, Fan)，向状态Fan广播事件event_1，由于状态Fan与子状态Fan.F1已激活，Fan.F1到Fan.F2的迁移有效，Fan.F2被激活。

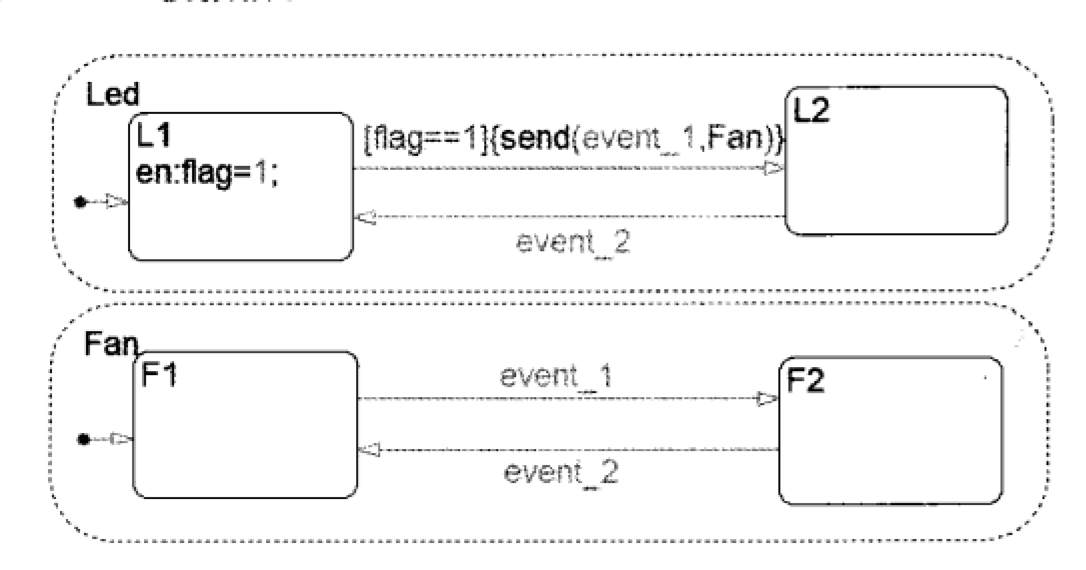


图3.5.6 send函数直接事件广播

2. 用事件名直接事件广播

将图3.5.6稍作修改，即可用事件名直接事件广播，如图3.5.8所示。

(1) 除事件event_1，另行定义Fan状态的本地事件event_1。

单击模型工具栏图标，在左侧model hierarchy区域中选择状态Fan，单击add→Event按钮，为Fan状态添加本地事件，如图3.5.7所示。同样的方法可以定义状态的本地数据。

(2) 将条件动作{send(event_1,Fan)}替换为迁移动作/Fan.event_1。

本例与用send函数直接事件广播的区别在于：

(1) event_1属于状态Fan的本地事件，作用范围限制在该状态内部，对Fan状态可见，对Led为不可见。

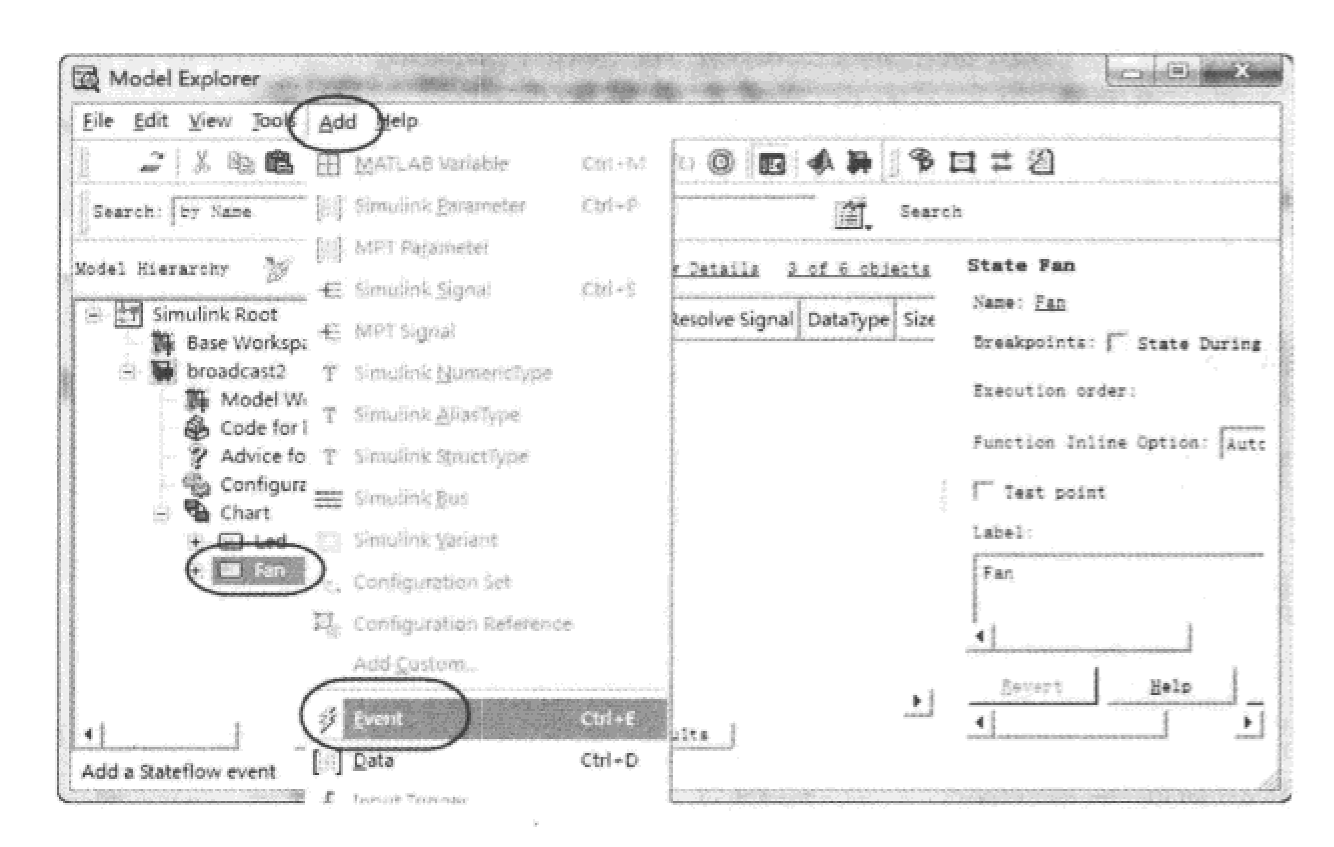


图 3.5.7　定义本地事件

(2) 迁移动作 Fan.event_1 替换了{send(event_1,Fan)}。

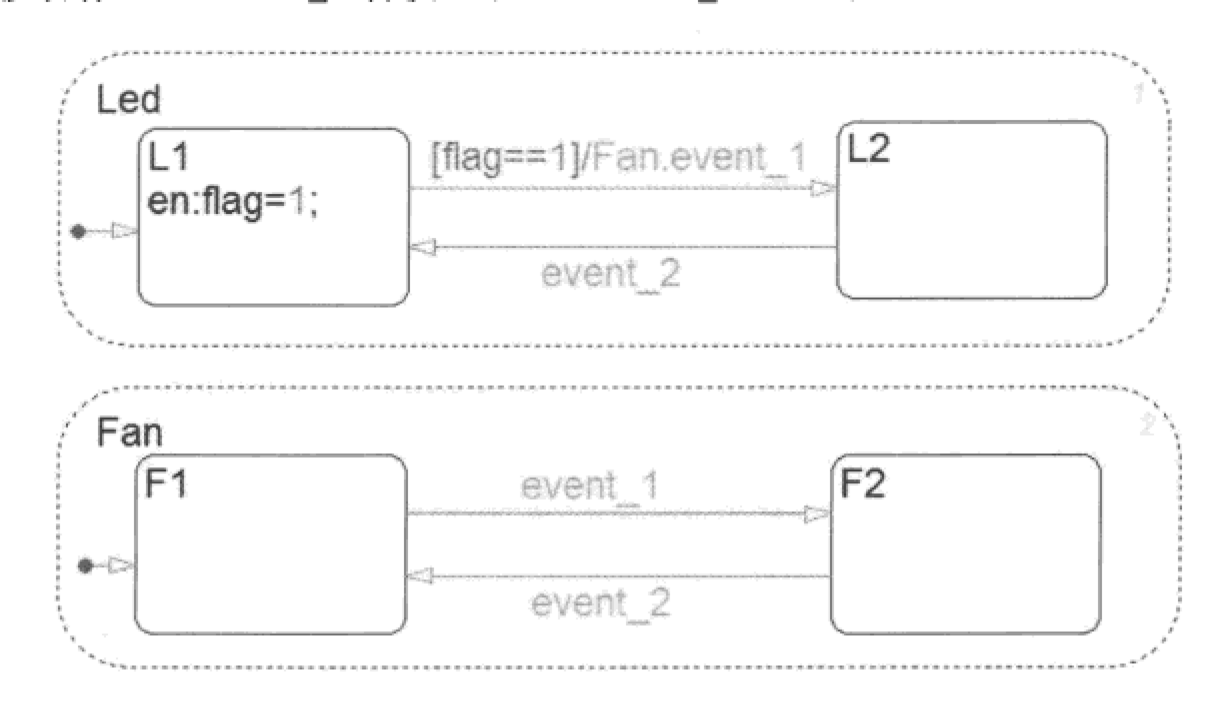


图 3.5.8　事件名直接事件广播

3.5.5　隐含事件和条件

隐含事件是一种内置事件，它不是由用户显式地定义或触发，而是当状态图执行时就会自动发生。例如状态图被唤醒、进入一个状态、退出一个状态或向内部数据对象赋值。隐含事件是它们发生时所在的状态的子对象，并且只对其父状态可见。

使用隐含事件和条件有助于简化并行状态之间的依赖关系，也可以减少数据字典中定义的事件数量，降低状态图的复杂程度。

表 3.5.1 列出了隐含事件/条件的表达式和对应的含义。

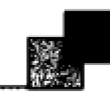


表 3.5.1　各隐含事件/条件及其含义

隐含事件/条件	含　义
change(data_name) or chg(data_name)	对指定变量(data_name)写入数据时，隐含地产生一个本地信号。该变量不能为 machine 的子数据，此隐含事件只对 Chart 或更低的层次有效。对于 machine 的子数据，用变化监测运算符决定其数据是否改变
enter (state_name) or en(state_name)	进入指定状态(state_name)时，隐含地产生一个本地信号
exit (state_name) or ex(state_name)	退出指定状态(state_name)时，隐含地产生一个本地信号
tick	评估动作所在的状态图被唤醒时，隐含地产生一个本地事件
wakeup	与 tick 相同
[in(state_name)]	当指定状态(state_name)处于激活状态，条件为真

如图 3.5.9 所示，Led 和 Fan 是两个并行状态，当 PowerOn 激活时，状态 Led. Off 和 Fan. Off 被激活。当隐含事件 tick 事件广播 6 次后，状态 Led. Off 向状态 Led. On 迁移，状态 Led. Off 退出，广播隐含事件 exit(Led. Off)，于是状态 Fan. Off 向状态 Fan. On 迁移。

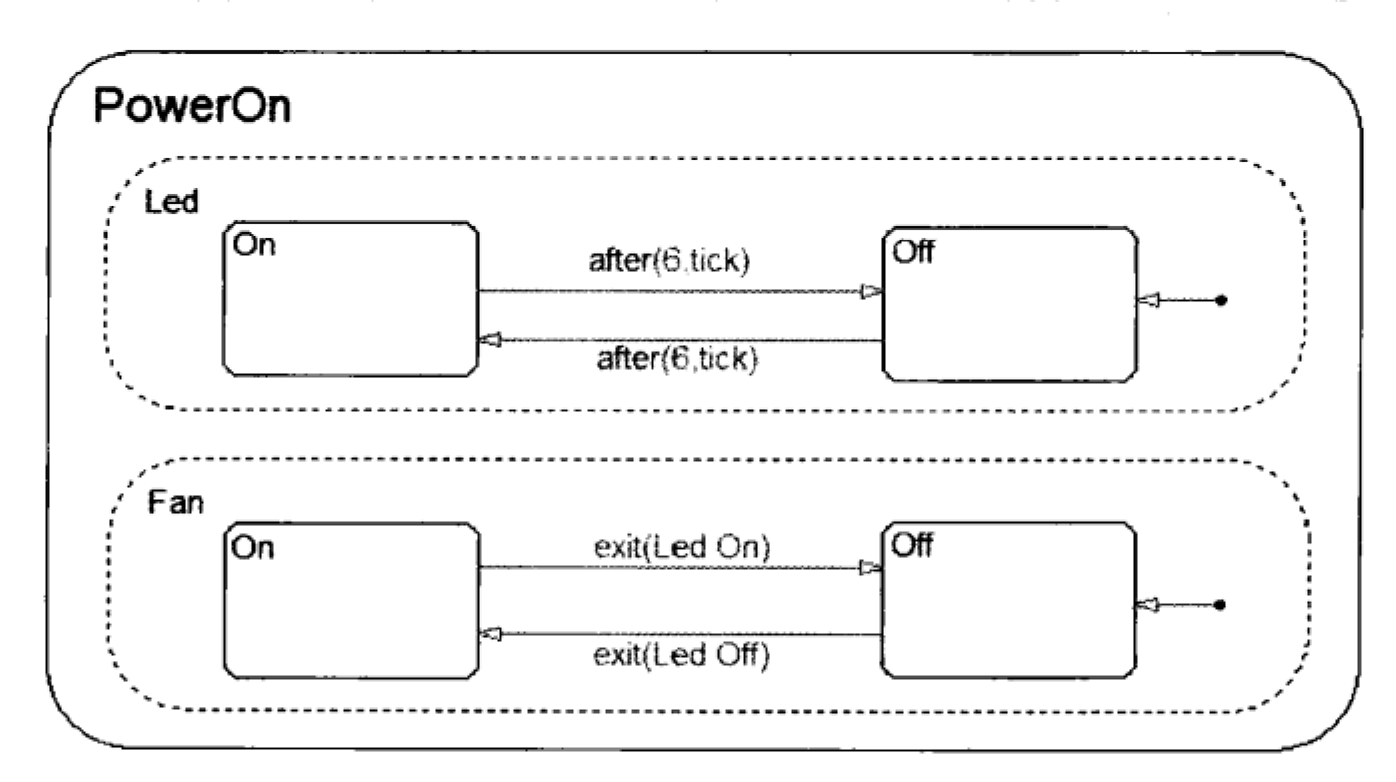


图 3.5.9　隐含事件的使用

3.6　Stateflow 其他对象

3.6.1　真值表(Truth table)

熟悉数字电路的用户，一定了解表 3.6.1 所列的异或门真值表。

表 3.6.1 异或门真值表

输入		输出	输入		输出
A	*B*	*Y*	H	H	L
H	L	H	L	H	H
L	L	L			

由于在 Stateflow 中，真值表的表达形式为条件，决策和动作，为了后面使用方便，可将异或门的真值表改写为表 3.6.2 的形式。

表 3.6.2 异或门的真值表改写形式

条件	决策 1	决策 2	决策 3	决策 4	决策 5(默认决策)
A==0&&B==0	T	—	—	—	—
A==0&&B==1	—	T	—	—	—
A==1&&B==0	—	—	T		—
A==1&&B==1	—	—	—	T	—
动作	y=0	y=1	y=1	y=0	y=−1

条件栏输入的每一个条件需判断真假，判断完结果为 T(逻辑真)，F(逻辑假)或一(逻辑真或逻辑假)。当所列的条件满足某一决策时，执行该决策所对应的动作，而动作的具体内容则另外在动作表中定义。默认决策定义了除决策 1～4 之外的所有情况。

将表 3.6.2 中的内容填写到 Stateflow 真值表中。需要注意的是动作的具体内容在 Action Table 中定义，并给每个动作赋予一个标号，在 Condition Table 的动作栏引用动作的标号即可，如图 3.6.1 所示。

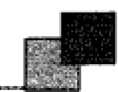
Condition Table

	Description	Condition	D1	D2	D3	D4	D5
1		a==0&&b==0	T	-	-	-	-
2		a==0&&b==1	-	T	-	-	-
3		a==1&&b==0	-	-	T	-	-
4		a==1&&b==1	-	-	-	T	-
		Actions: Specify a row from the Action Table	A1	A2	A2	A1	A3

Action Table

#	Description	Action
1		A1:y=0
2		A2:y=1
3		A3:y=-1

图 3.6.1 建立真值表



在每一个仿真步长内，系统首先逐一判断各个条件的输出结果，将输出结果组合起来，再与各个决策逐一比较，当结果完全满足某一决策时，即执行对应的动作，同时不再判断后续的决策，如图 3.6.2、图 3.6.3 所示。

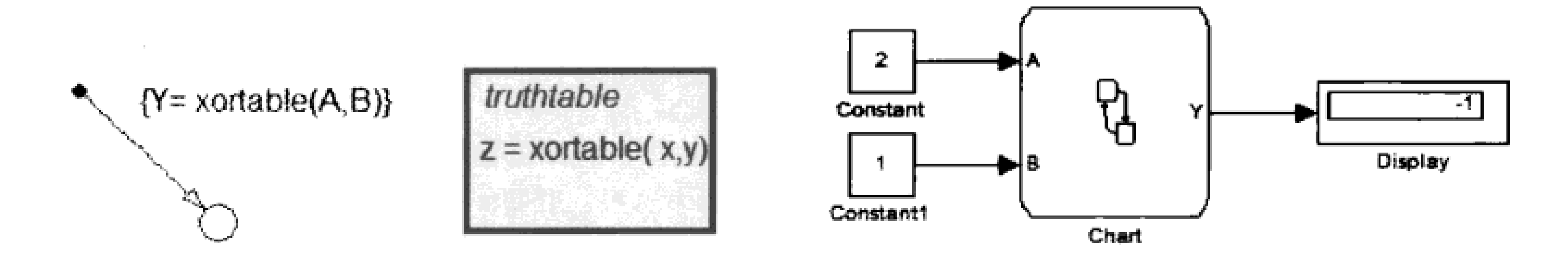


图 3.6.2　Stateflow 流程图　　　　图 3.6.3　Simulink 模型

3.6.2　图形函数(Graphical function)

图形函数是用包含 Stateflow 动作的流程图定义的函数，是流程图的延伸，使用图形方式定义算法，并在仿真过程中跟踪它的运行。

图形函数与 MATLAB 函数、C 函数有一定的相似之处，例如：图形函数同样需要接收参数并返回结果；用户可以在状态/迁移动作中调用图形函数。不同之处在于图形函数是 Stateflow 自身的图形对象，因此可以直接通过 Stateflow 编辑器创建并调用，不必像文本函数一样需要用外部工具创建，在外部定义。

下面将通过图形函数完成异或的功能，使读者了解图形函数的用法。

(1) 先创建一个包含 Stateflow 图表的 Simulink 模型，然后单击图表编辑器中图形工具栏的图形函数按钮，即可在编辑器空白位置添加图形函数，如图 3.6.4 所示。

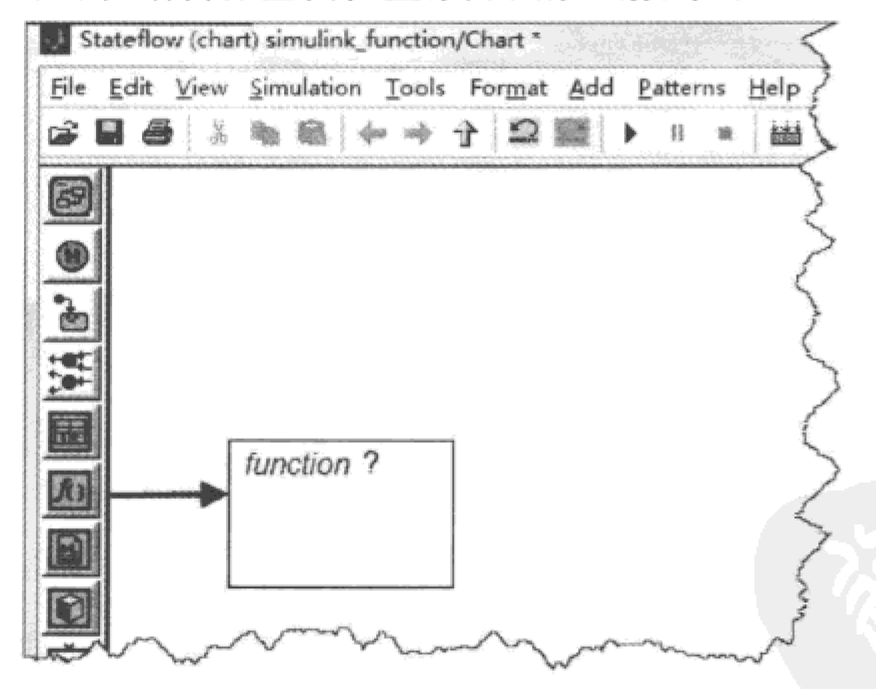


图 3.6.4　添加图形函数

(2) 单击？处进入编辑状态，用户可以在此定义所需要的函数。函数名的语法规则为：[n1,n2,n3…]＝函数名(a1,a2,a3…)，其中 n1,n2,n3 为返回值，a1,a2,a3 为参数值，如图 3.6.5所示。

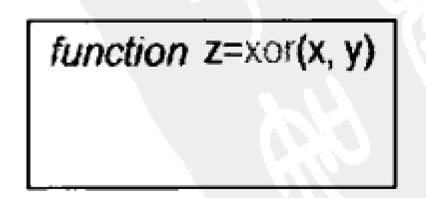


图 3.6.5　定义函数

(3) 用户可以在图形函数内部用图形对象完成所需的函数逻辑。在本例中使用 Stateflow 提供的 if-elseif-else … 模板完成函数，使问题更加简单。右击图形函数，选择 Pattern→Add Decision→If-Elseif-Else … 命令，如图 3.6.6 所示。

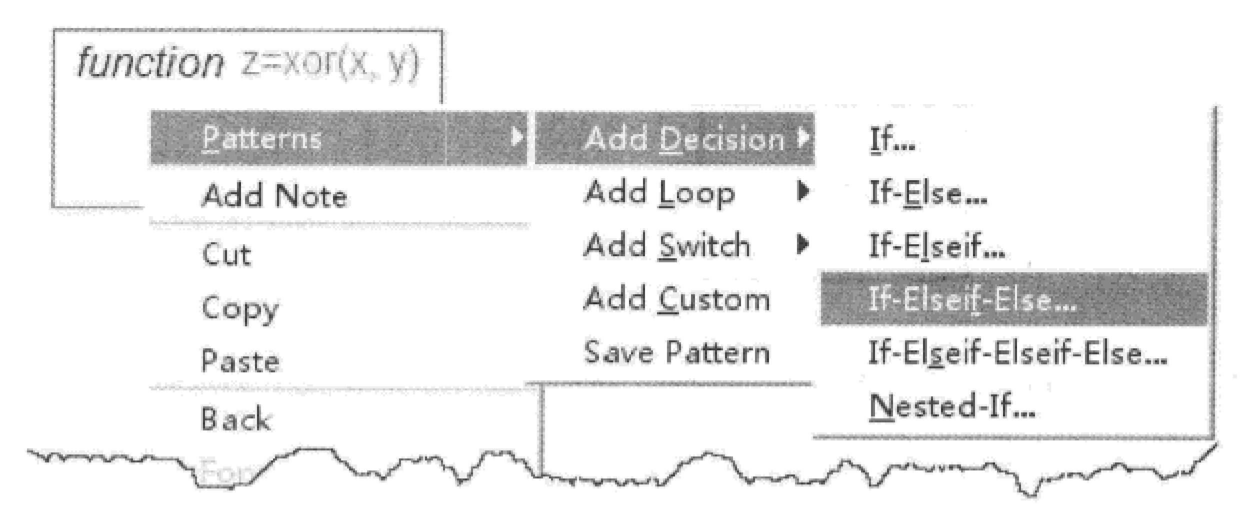


图 3.6.6　选择逻辑图样模板

(4) 根据 3.6.1 节的内容不难得出异或函数的实现逻辑。在 Stateflow Pattern：IF-ELSEIF-ELSE 对话框中完成异或逻辑，如图 3.6.7 所示。

(5) 单击 OK 铵钮确认后，自动生成实现该函数的图形函数，如图 3.6.8 所示。

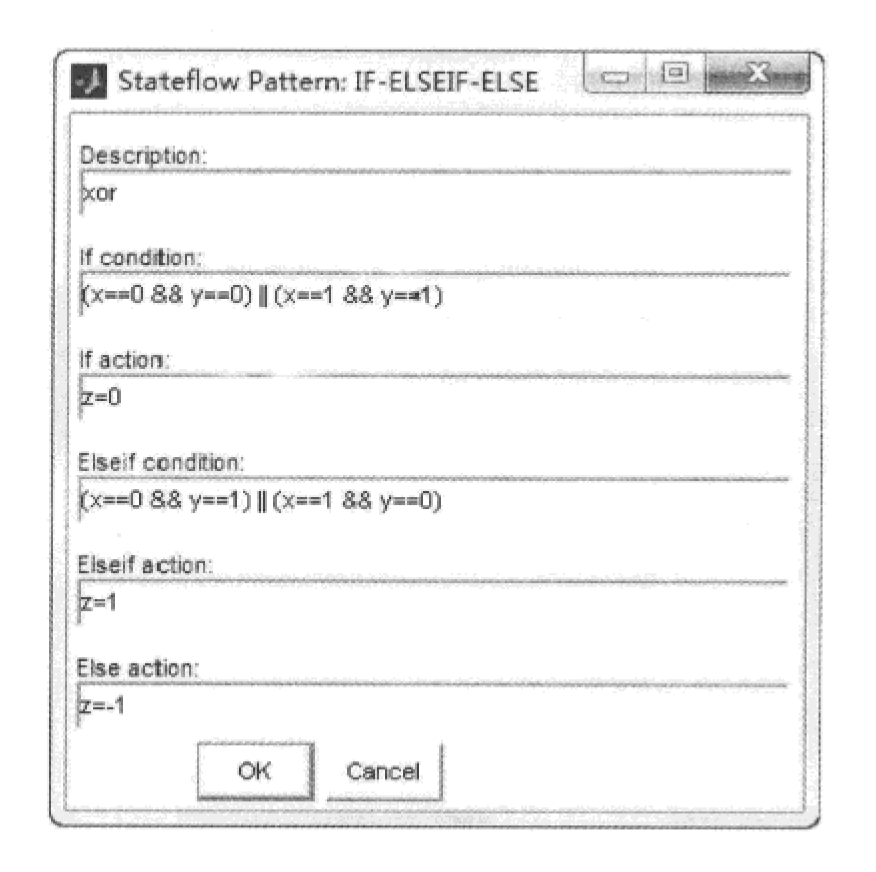


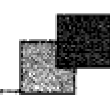
图 3.6.7　流程图对话框

[(x==1&&y==1)||(x==0&&y==0)]
{z=0;}
[(x==1&&y==0)||(x==0&&y==1)]
{z=1;}
{z=-1;}

图 3.6.8　流程图

由于该图形函数较复杂，可以先将图形完全包含在函数框中，如图 3.6.9 所示，然后用右键菜单中的 Make Contents→subcharted 功能简化图形函数，如图 3.6.10 所示。

(6) 图表添加相应的数据、事件后，就可以通过状态动作或迁移动作实现图形函数的调用了，如图 3.6.11 所示。



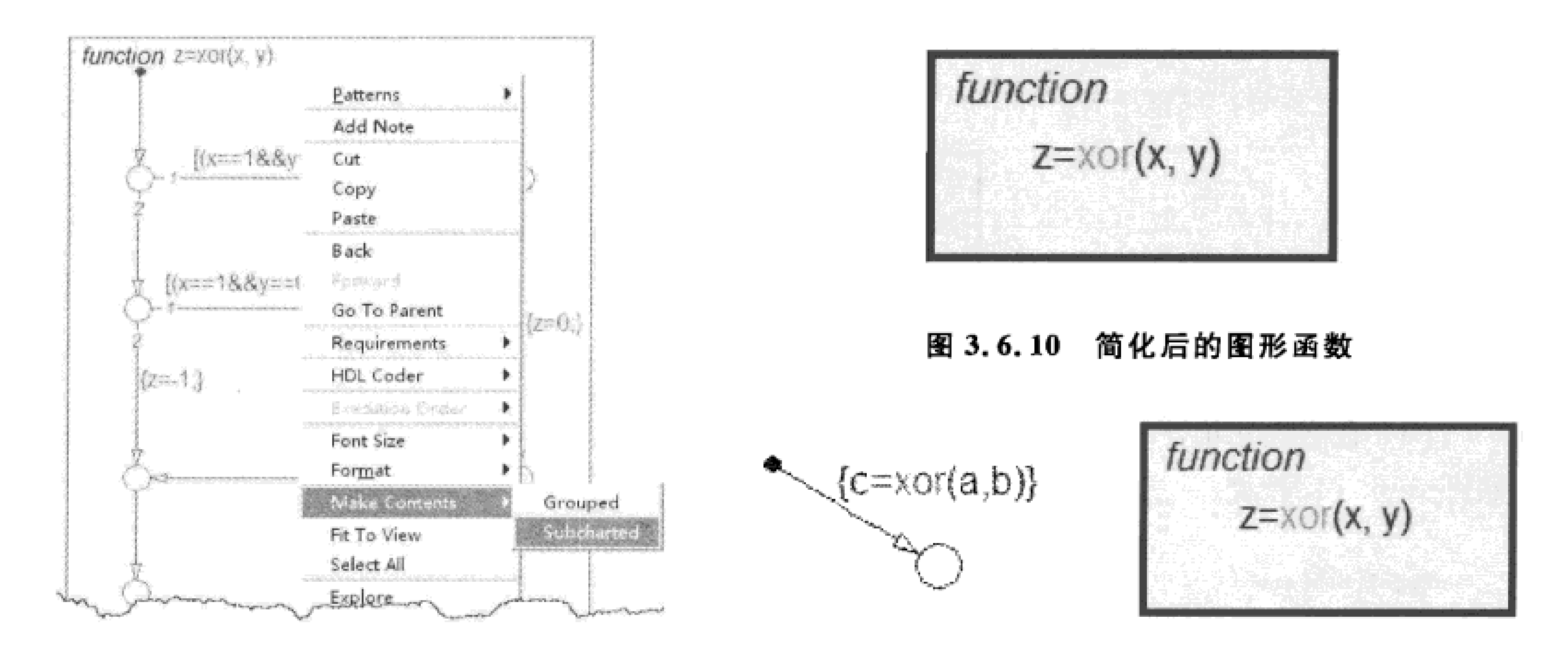


图 3.6.9　简化图形函数

图 3.6.10　简化后的图形函数

图 3.6.11　调用 Stateflow 图形函数

3.6.3　Embedded MATLAB

Embedded MATLAB 函数可以为 Stetaflow 添加 MATLAB 函数。在描述算法代码方面，文本形式的 MATLAB 语言比图形化的 Stateflow 动作语言更加优越，因此，必要时可使用 Embedded MATLAB 函数对象，在 Stateflow 状态图里添加 MATLAB 函数。

Embedded MATLAB 函数是 MATLAB 语言的一个子集，该子集能够有效优化生成代码，提高代码效率，为编译目标生成产品级 C 代码。Embedded MATLAB 函数可以调用子函数、Embedded MATLAB 运行时库函数、Stateflow 函数以及部分 MATLAB 函数和定点工具箱运行时库函数。

(1) 创建一个包含 Stateflow 图表的 Simulink 模型，然后单击图表编辑器中图形工具栏的 Embedded MATLAB 函数按钮，即可在编辑器空白位置添加 Embedded MATLAB 函数，如图 3.6.12 所示。

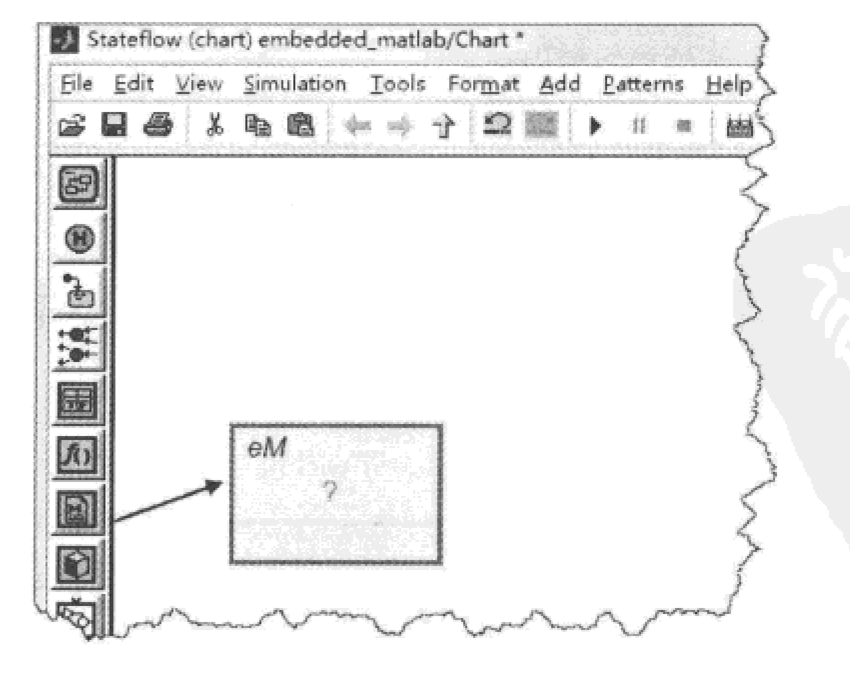


图 3.6.12　添加 Embedded MATLAB 函数

(2) 单击"?"处可以编辑 Embedded MATLAB 函数名称,命名规则与图形函数类似,此处命名为[len,area]=rectangle(length,width),如图 3.6.13 所示。



图 3.6.13 Embedded MATLAB 函数命名

(3) 在 Stateflow 中定义 4 个数据对象:length、width 为函数输入,len、area 为函数输出,如图 3.6.14 所示。

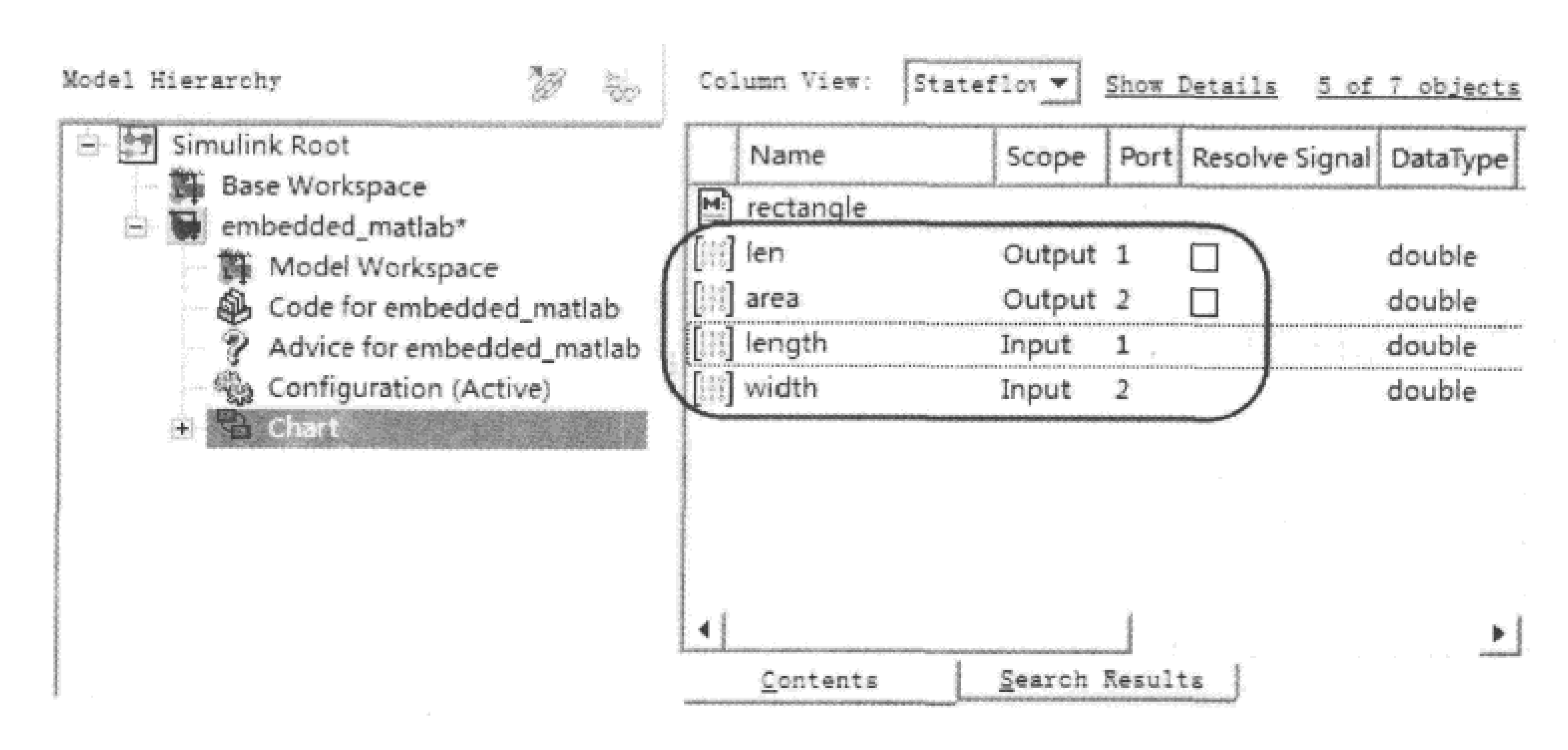


图 3.6.14 定义数据对象

(4) 双击 Embedded MATLAB 函数对象,打开 Embedded MATLAB 函数编辑器,此时函数定义行已定义,只需添加算法代码即可。

```
function [len,area] = rectangle(length,width)
% #eml
len = 2 * (length + width);  % 计算矩形周长
area = length * width;       % 计算矩形面积
end
```

关于 Embedded MATLAB 的编写与调试在第 1 章有详细的介绍,这里不再赘述。

(5) 在图表中添加默认迁移和终止节点,通过迁移动作调用 Embedded MATLAB 函数,如图 3.6.15 所示。

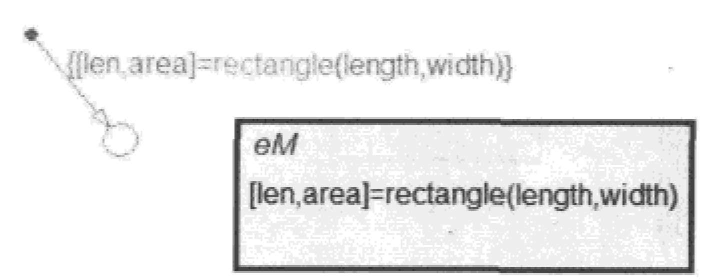
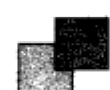


图 3.6.15 调用 Embedded MATLAB 函数



3.6.4 图形盒(Box)

图形盒是 Stateflow 的一种图形对象，可以用来组织图表中的图形对象。添加图形盒后，图形盒中图形对象的可视性以及并行状态的激活顺序都会有所变化。

引入图形盒可以为图表新增一个层次，因此，对于图形盒外部的对象来说，图形盒内部对象的可见性发生了变化。用户若需要在图形盒外部引用位于图形盒内部的函数或状态，则应该在引用路径中添加图形盒的名称。

引入图形盒会改变并行状态的激活顺序。图形盒内部并行状态激活顺序为从上到下，从左到右；若图形盒外部也存在并行状态，则图形盒内部的状态优先于外部状态激活。

图形盒的使用有如下规则：

(1) 若在图形盒外部引用图形盒内部的函数或状态，必须在路径上加入图形盒的名称。

(2) 在图形盒内部可以添加图形对象，如函数、状态。

(3) 内部包含若干对象的状态可以转换为图形盒。

(4) 为图形盒添加数据对象可以使其内部所有对象共享该数据。

(5) 图形盒及其内部对象可以打包称为一个图形对象。

(6) 图形盒可通过 subchart 功能简化其内部元素。

(7) 图形盒无法定义状态动作，如 entry、during 和 exit 动作。

(8) 无法定义始于或终于图形盒的迁移。

在 Stateflow 编辑窗口中单击图形盒按钮，再单击编辑窗口中的空白处即可添加图形盒。单击?处编辑图形盒名称，如图 3.6.16 所示。

如图 3.6.17 所示，图形盒命名为 status，其中有两个 Embedded MATLAB 函数，若要调用 msgCold，其语法为 Status. msgCold()，若要调用 msgWarm，其语法为 Status. msgWarm()。

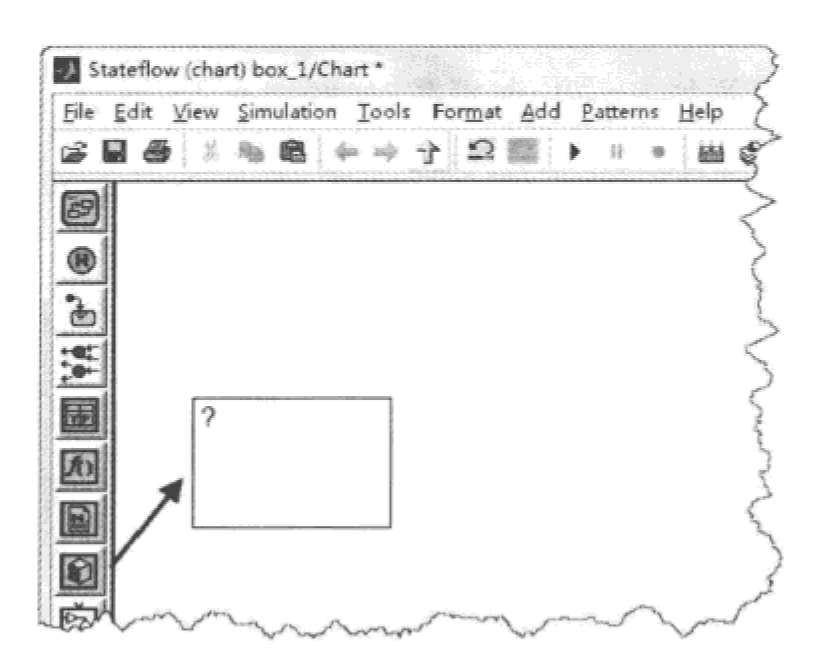


图 3.6.16 添加图形盒

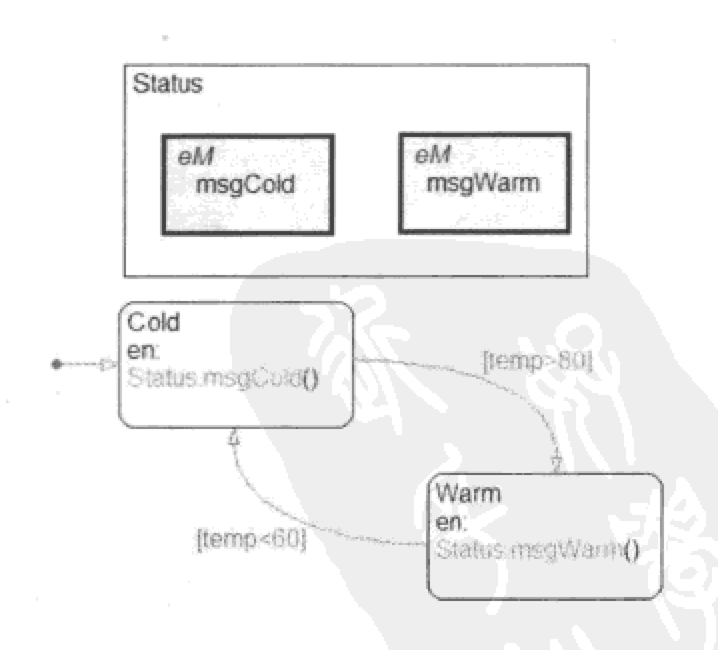


图 3.6.17 调用图形盒中的对象

3.6.5 Simulink 函数调用

Simulink 函数是 Stateflow 图表中的一种图形对象,用户可以在该对象中添加 Simulink 模块来实现预期的功能,并在状态/迁移动作中调用,从而实现了 Simulink 与 Stateflow 间的无缝连接。Simulink 函数通过减少图形和非图形对象的方式,使模型设计更加高效,更具可读性。

Simulink 函数一般应用于以下两方面:

(1) 需要用到 lookup tables 等 Simulink 模块的函数。

(2) 多控制器的调度执行。

Simulink 函数的调用与 Simulink 模型中的函数调用子系统模块类似,但是有表 3.6.3 所列出的区别。

表 3.6.3 两种函数调用的差别

行　为	函数调用子系统	Simulink 函数
执行中是否需要函数调用输出事件	是	否
是否需要信号线	是	否
是否支持基于帧的输出信号	是	否

Simulink 函数的定义位置则决定了其影响范围,如表 3.6.4 所列。

表 3.6.4 Simulink 函数作用范围

Simulink 函数定义于	调用 Simulink 函数的范围	Simulink 函数定义于	调用 Simulink 函数的范围
状态	该状态本身及其子状态	图表	该图表内皆可调用

以下说明 Simulink 函数的用法。

(1) 创建一个包含 Stateflow 图表的 Simulink 模型,然后单击图表编辑器中图形工具栏的 Simulink 函数按钮,即可在编辑器空白位置添加 Simulink 函数,如图 3.6.18 所示。

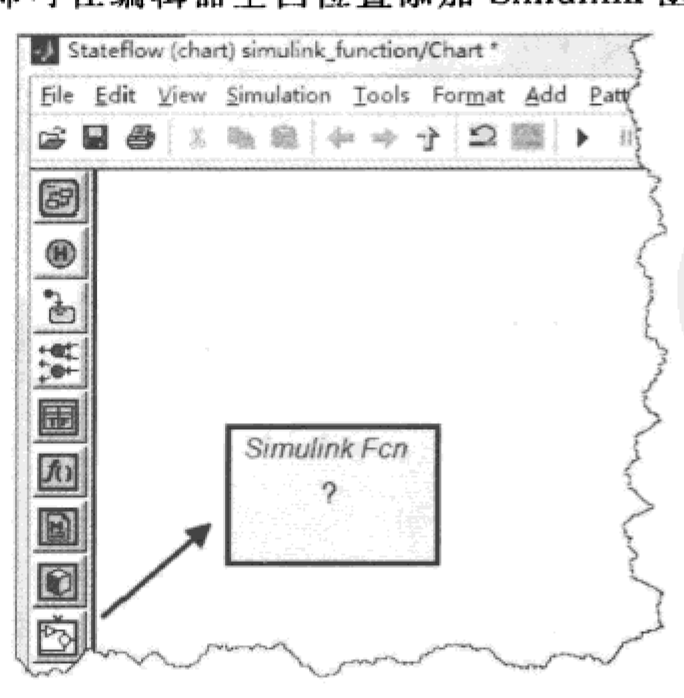
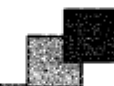


图 3.6.18 添加 Simulink 函数



（2）单击？处进入编辑状态，用户可以在此定义所需要的函数。函数命名形式与图形函数相同。此处将 Simulink 函数命名为 x=sim_fun(a,b)，如图 3.6.19 所示。

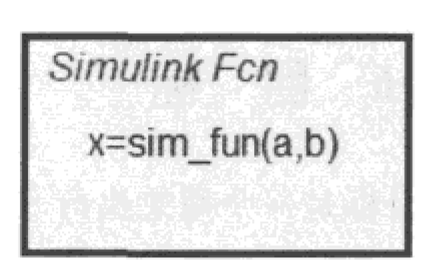


图 3.6.19　定义函数

（3）双击 Simulink 函数对象，定义 Simulink 函数的子系统元素。在子系统中有与命名的函数对应的输入/输出端口和一个函数调用触发端口，如图 3.6.20 所示。

向模型中添加模块，完成图 3.6.21 所示的子系统模型。注意应保留模型中的触发端口。

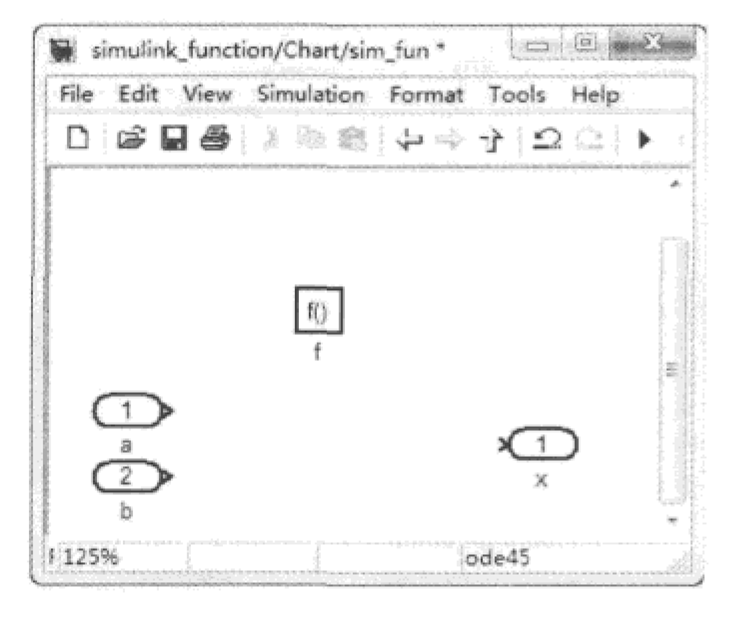


图 3.6.20　打开 Simulink 函数

图 3.6.21　添加模块

（4）定义两个 input from Simulink 属性的数据对象作为函数的输入；定义一个 output to Simulink 属性的数据对象作为函数的输出，如图 3.6.22 所示。

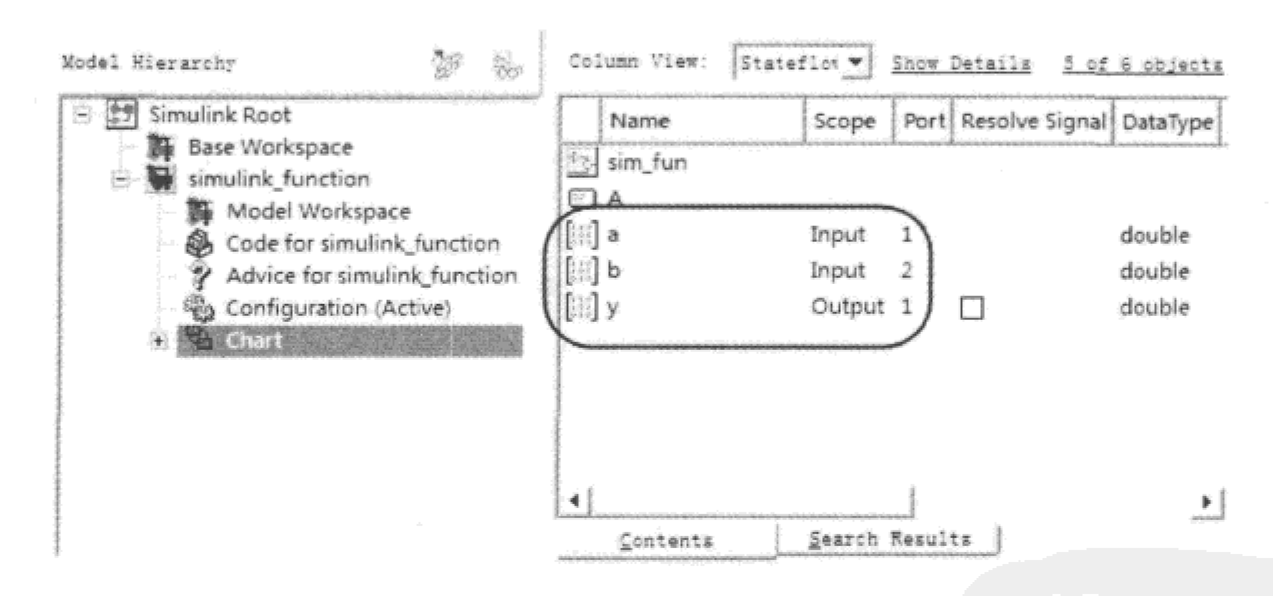


图 3.6.22　定义数据对象

（5）向图表中添加状态，并定义其 entry 动作为 sim_fun(m,n)，则当该状态被激活时，即调用 Simulink 函数 sim_fun()，如图 3.6.23 所示。

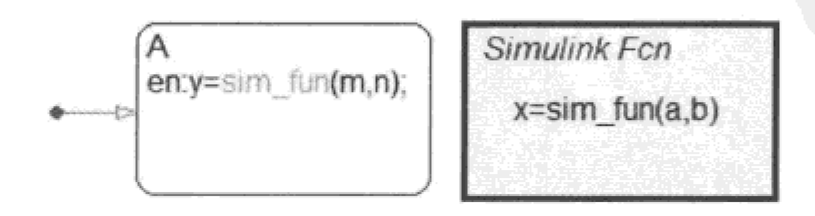


图 3.6.23　调用 Simulink 函数

3.6.6 目　标

目标是一个用来执行 Stateflow 图表或包含 Stateflow 状态机的 Simulink 模型的程序。

若用户并不想为具体应用建立快速原型或产品，而是要生成其独立代码，则可以使用代码生成软件 Stateflow Coder 建立用户目标，但该软件生成的代码并未经过 RTW 的优化。

(1) Stateflow 编辑窗口单击 add→targets 会弹出用户目标对话框如图 3.6.24 所示。

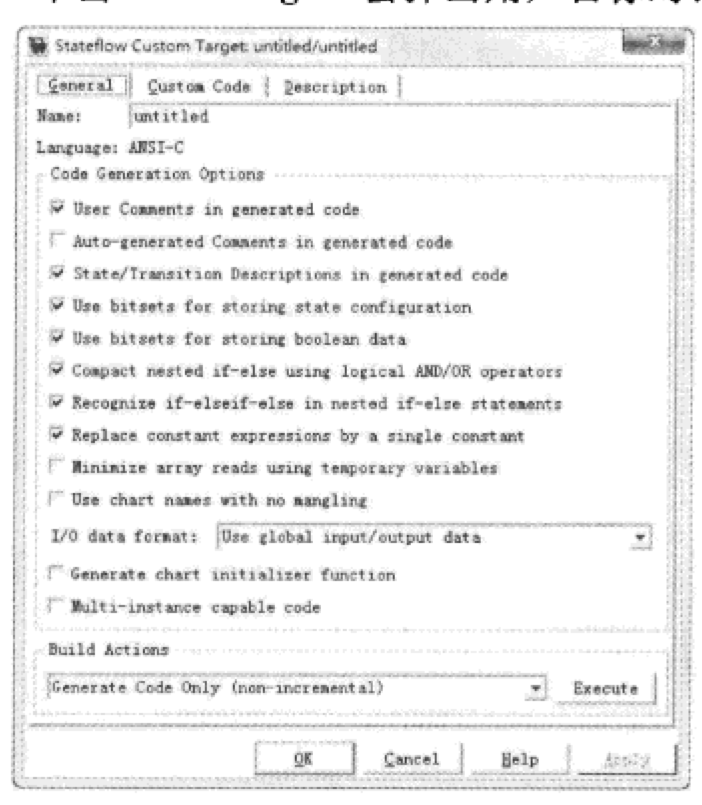


图 3.6.24　用户目标对话框

在 Name 区域内输入目标名称并设置完毕后，单击 OK 按钮即完成用户目标的添加。

(2) 若用户需要进一步修改对目标的配置，可以单击按钮▣，在模型浏览器中进行操作。

在模型浏览器的 Model Hierarchy 面板内选择含有用户目标的主模型，然后在 Contents 面板中选择目标，则右侧的动态面板即出现用户目标的配置对话框，如图 3.6.25 所示。

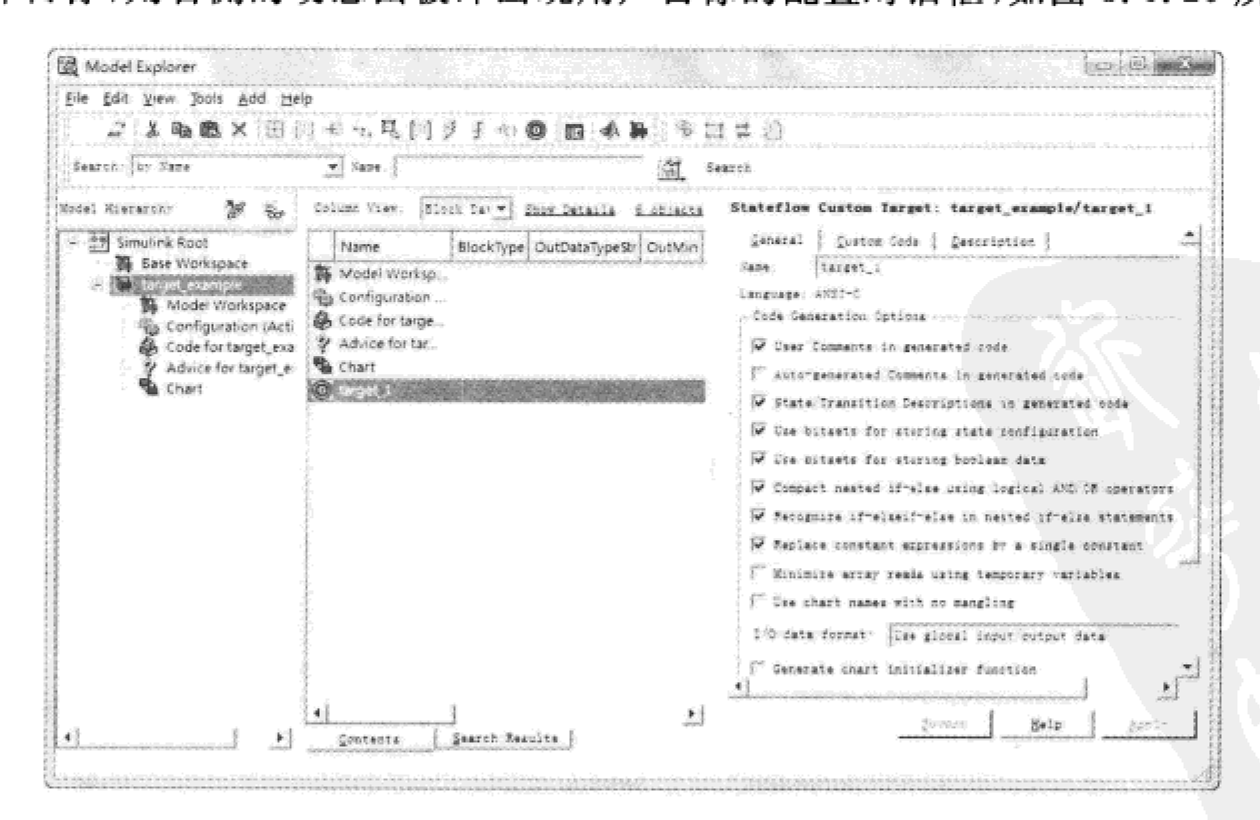
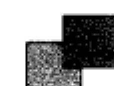


图 3.6.25　用户目标的配置对话框



(3) 用户目标配置对话框的 General 面板内可做如下设置:

① User Comments in generated code:在生成代码中包含用户自定义的注释。

② Auto-generated Comments in generated code:在生成代码中包含自动生成的注释。

③ State/Transition Descriptions in generated code:在生成代码中包含状态和迁移的描述。

④ Use bitsets for storing state configuration:减少存储变量的内存用量。但是当目标处理器不包含操作 bitsets 的说明时,能够增加存储目标代码的内存用量。

⑤ Use bitsets for storing boolean data:减少存储布尔型变量的内存用量。但是当目标处理器不包含操作 bitsets 的说明时,能够增加存储目标代码的内存用量。

⑥ Compact nested if-else using logical AND/OR operators:通过使用逻辑运算符压缩多层嵌套的 if-else 结构来提高代码的可读性。

例如,生成代码:

```
if(c1) {
    if(c1) {
        a1();
    }
}
```

被表示为:

```
if(c1 && c2) {
a1();
}
```

⑦ Recognize if-else if-else in nested if-else statements:通过使用 if-else if-else 结构提高多层次嵌套的 if-else 代码的可读性。

例如,生成代码:

```
if(c1) {
    a1();
}else{
    if(c2) {
        a2();
    }else{
        if(c3) {
            a3();
        }
    }
}
```

被表示为:

```
if(c1) {
    a1();
```



```
}else if(c2) {
    a2();
}else if(c3) {
    a3();
}
```

⑧ Replace constant expressions by a single constant:通过对常量表达式估值,用单个常量代替常量表达式来提高代码可读性。此优化功能还能消除无效代码。

例如,生成代码:

```
if(2 + 3 < 2) {
    a1();
}else {
    a2(4 + 5);
}
```

被表示为:

```
if(0) {
    a1();
}else {
    a2(9);
}
```

⑨ Minimize array reads using temporary variables:在条件允许时,使用临时变量,简化数组的读取。

例如,生成代码:

```
a[i] = foo();
if(a[i]< 10 && a[i]< 1) {
    y = a[i] + 5;
}else{
    z = a[i];
}
```

被表示为:

```
a[i] = foo();
temp = a[i];
if(temp < 10 && temp>1) {
    y = temp + 5;
}else{
    z = temp;
}
```

⑩ Use chart names with no mangling:保留图表 entry 函数名,以便用户可以在手写 C 代码中调用。勾选该复选框时,生成代码并不通过 mangle 图表的名称来使其唯一化。由于该复



选框不检测生成代码中的名称冲突性，因此只有当模型中图表名称唯一时可用。

⑪ I/O data format：

a. Use global input/output data ：生成图表的输入/输出数据并作为全局变量。

b. Pack input/output data into structures：为图表输入/输出数据生成结构。

⑫ Generate chart initializer function：生成数据初始化函数。

⑬ Multi-instance capable code：生成多实例图形对象代替静态定义。

⑭ Build Options：

a. Generate Code Only (non-incremental) ：为模型中的所用图表生成代码。

b. Rebuild All (including libraries) ：重建目标。若用户建立目标后更换了编译器或更新了目标文件，可选择该选项。

c. Make without generating code：调用创建程序，但并不生成代码。

3.7　综合应用

3.7.1　计时器

1. Stateflow 状态图

在图 3.3.2 的基础上，添加默认迁移的事件与条件动作，每当事件 TIC 发生，执行{percent = percent +1 }，之后再判断是否进行分秒进位，此即可作为一个计时流程。事件 TIC 可由脉冲模块模拟，触发条件为上升沿，如图 3.7.1 所示。

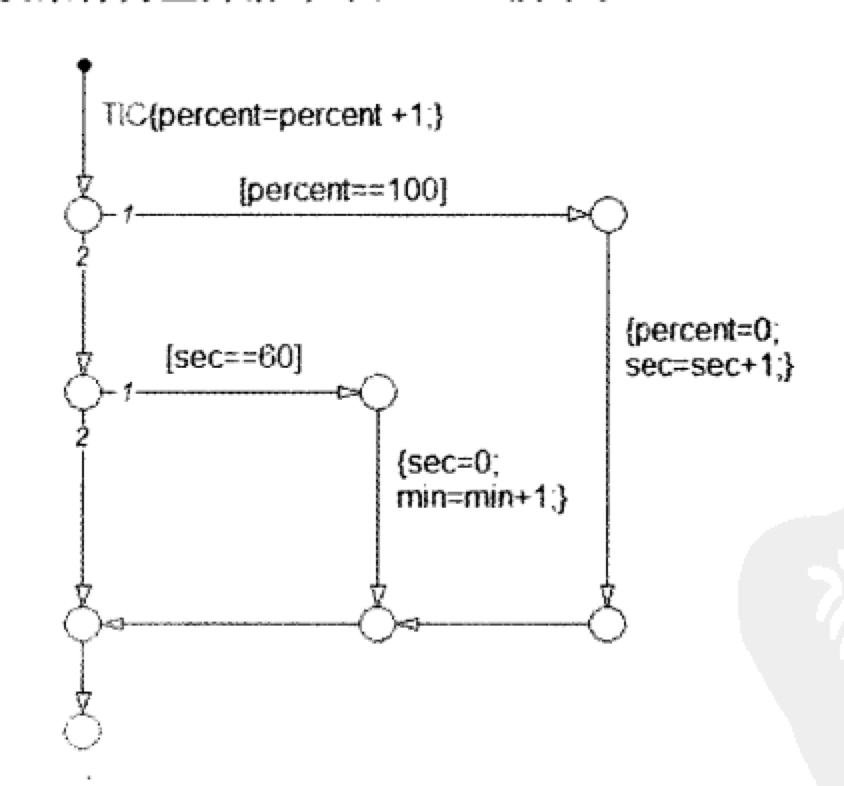
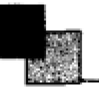


图 3.7.1　时钟流程图

为了实现以圈计时的功能，将图 3.7.1 的默认迁移起点，移到父状态 Run，这表示只要 Run 处于激活状态，就不断执行计时流程，不受 LAP 状态的影响，如图 3.7.2 所示。



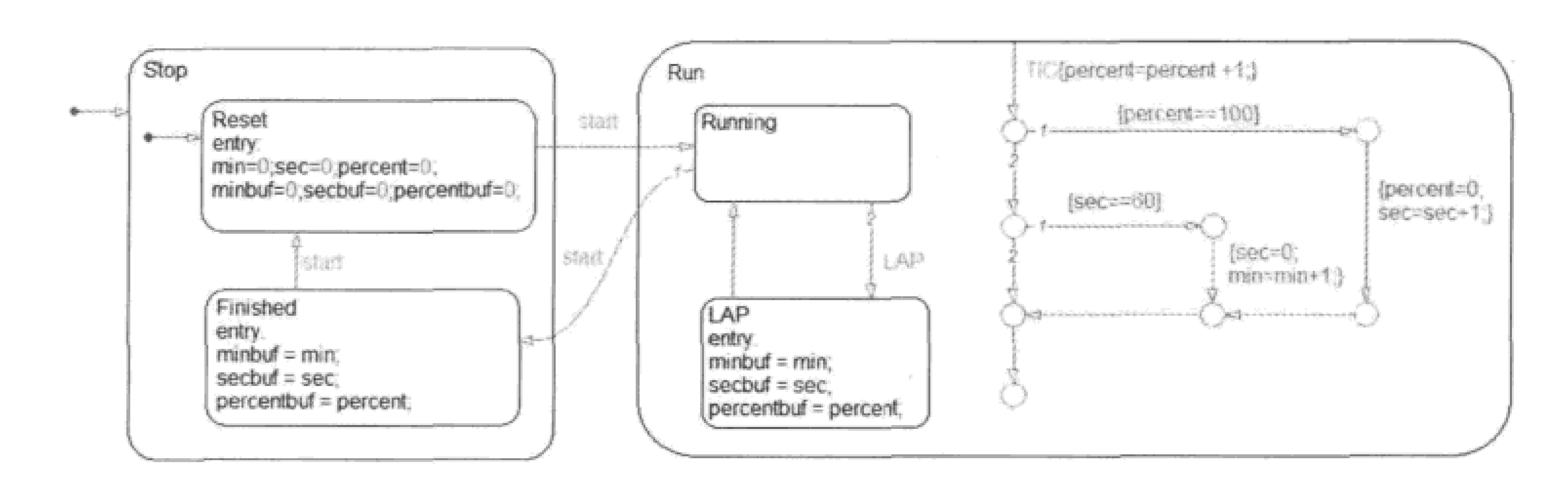


图 3.7.2　计时器状态图

2. Simulink 模型

以脉冲模块模拟输入时钟信号，两个开关分别模拟 Start 与 LAP 按钮，按图 3.7.8 建立 Simulink 模型。由于每个 Stateflow 模块只能有一个事件输入口，有多个外部输入事件时，必须使用 Mux 模块将它们组合成向量。水平放置的 Mux 模块，其输入端口号从左往右对应事件端口号，而垂直放置的 Mux 模块，其对应关系则是从上到下。

在 Simulink 模块库中找到图 3.7.3～图 3.7.7 所示模块，并按图 3.7.8 所示连接。

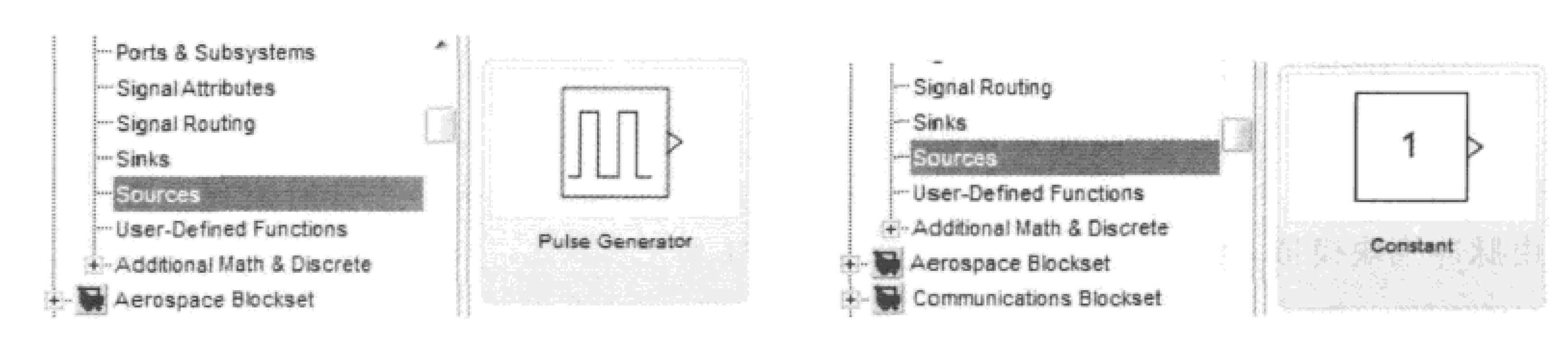


图 3.7.3　脉冲发生器模块　　　　图 3.7.4　常数模块

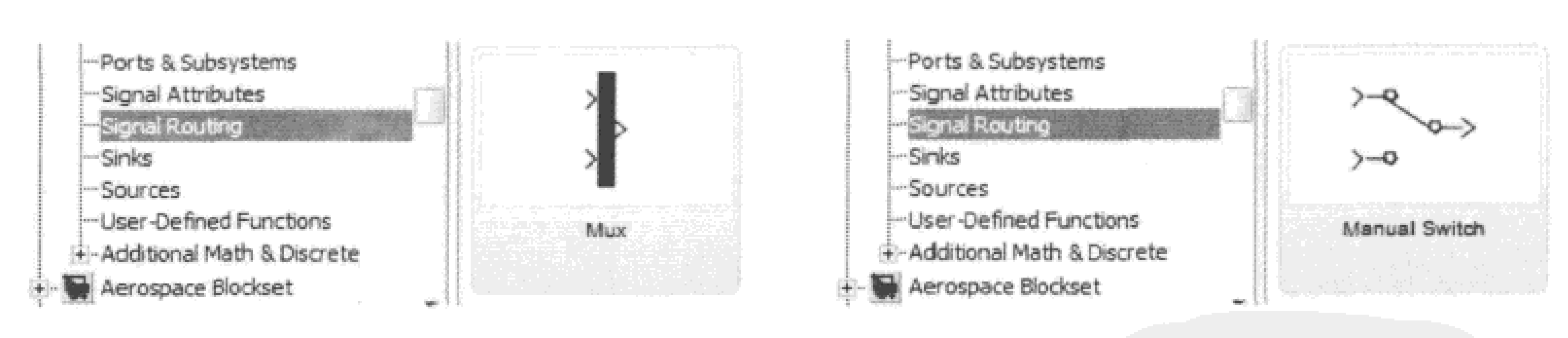


图 3.7.5　合路器模块　　　　图 3.7.6　手动开关模块

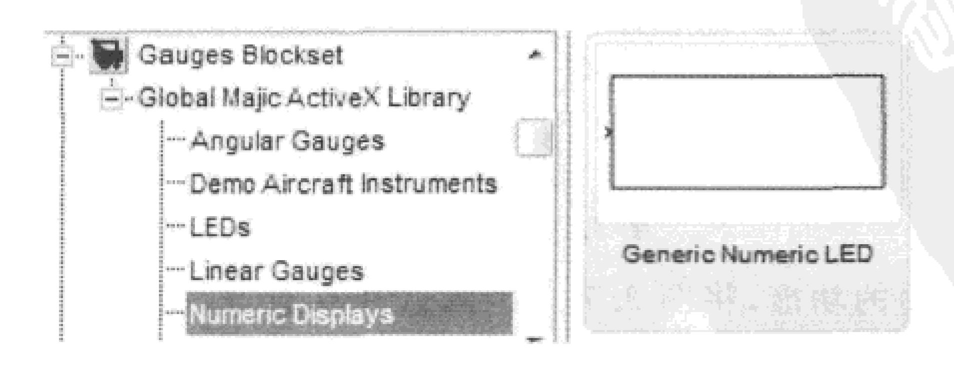
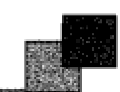


图 3.7.7　数字 LED 模块



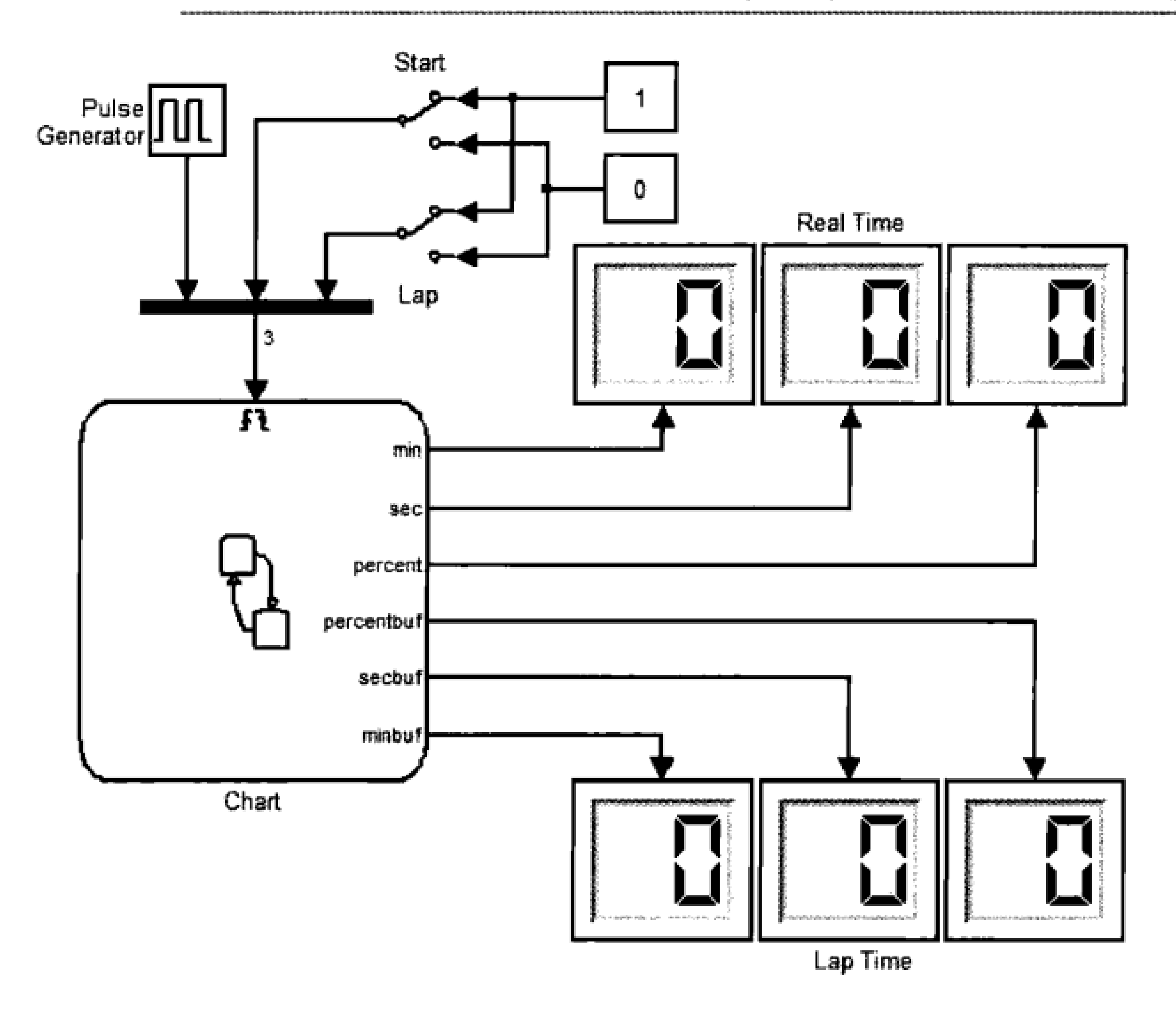


图 3.7.8　功能验证模型

双击 Start 开关，系统开始计时，如图 3.7.9 所示。

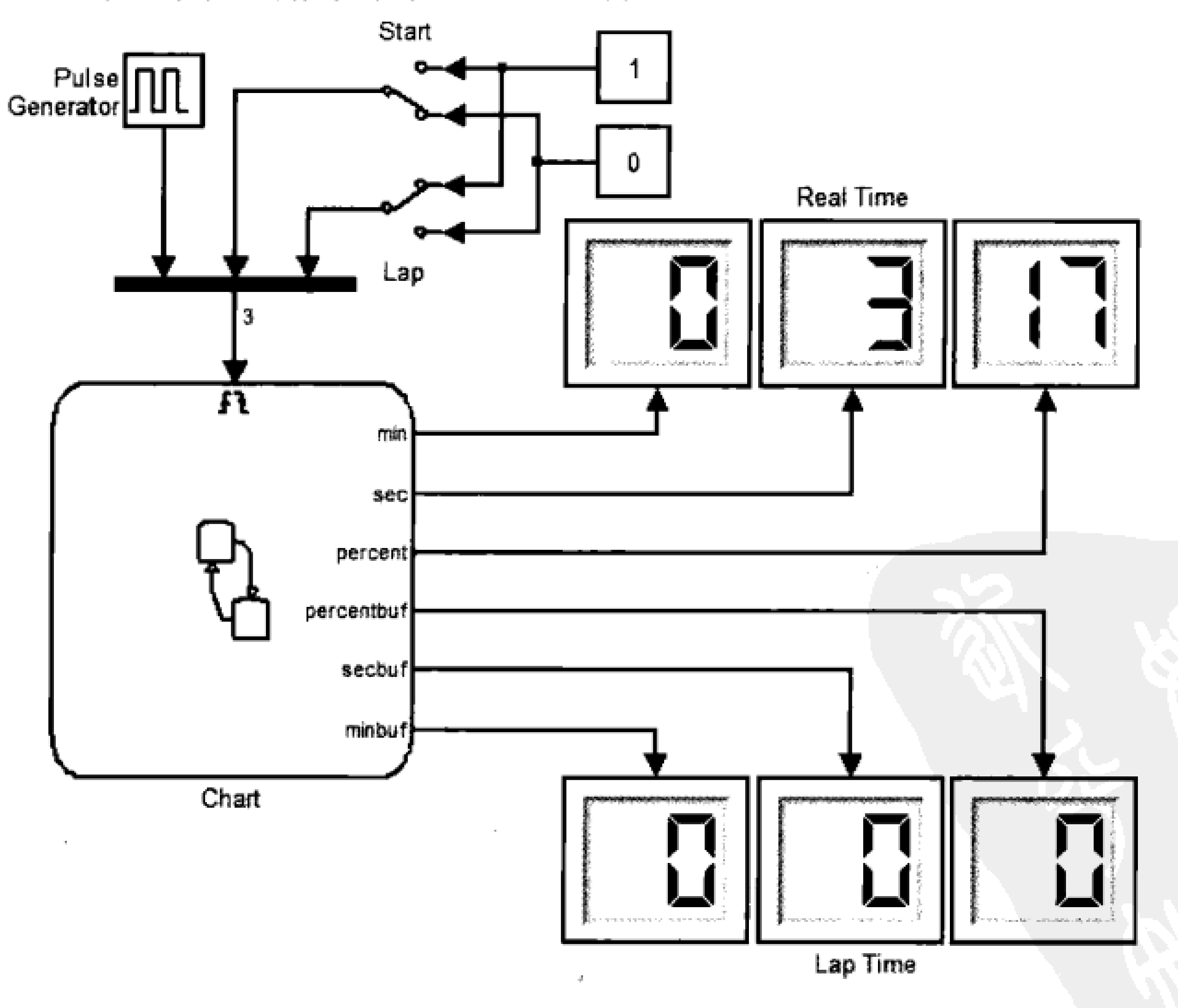


图 3.7.9　开始计时

双击 Lap 开关，系统记录当前计时值，并继续计时，如图 3.7.10 所示。

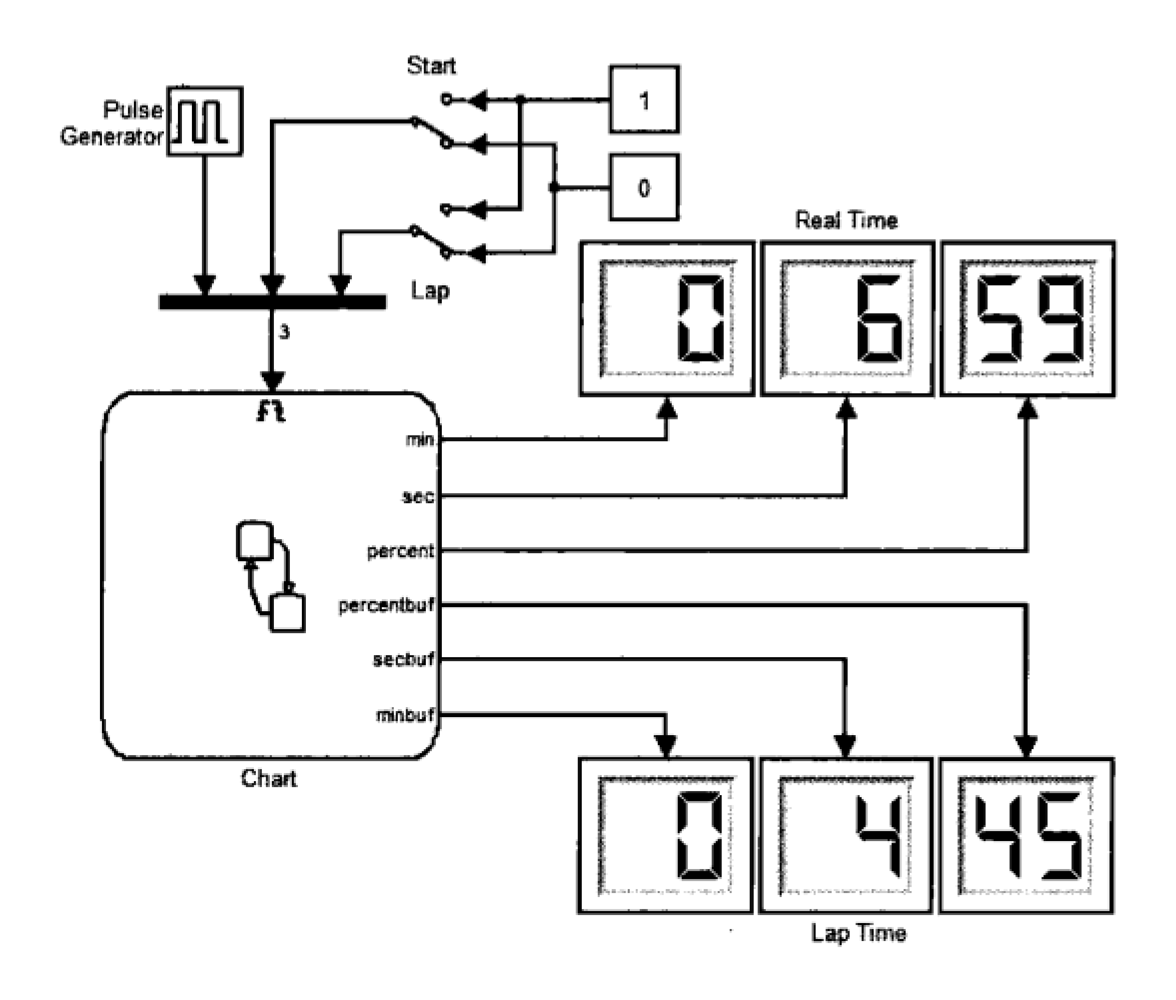


图 3.7.10　以圈计时

再次双击 Start 开关，系统停止计时，显示最后计时值，如图 3.7.11 所示。

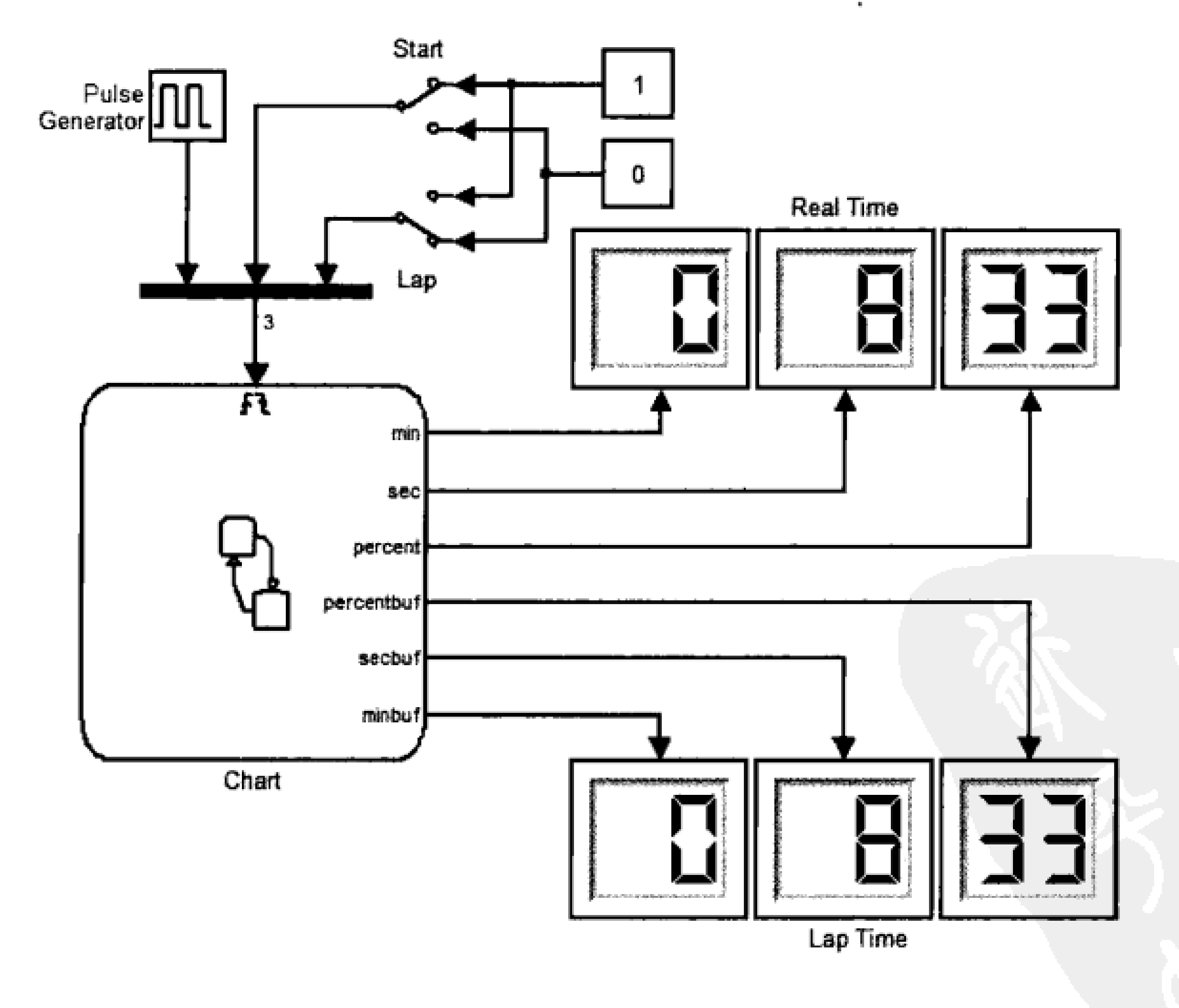
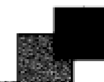


图 3.7.11　最后计时值



3. 创建 GUI 界面

利用 Simulink 模块库提供的开关模块，已能实现本例的功能，但若模型需要大量的开关，必然导致常数模块或连线间的交叉点增多，人为地使得模型变得复杂。为此使用 GUI 用户图形界面，简化模型，同时提高仿真的舒适度。

MATLAB 的 GUI 界面与 VC、VB 等软件的类似，因此本文仅叙述创建过程，不作功能介绍。

(1) 删除图 3.7.8 模型的开关以及常数模块，另外新增常数 Start 与 Lap，按图 3.7.12 所示调整。

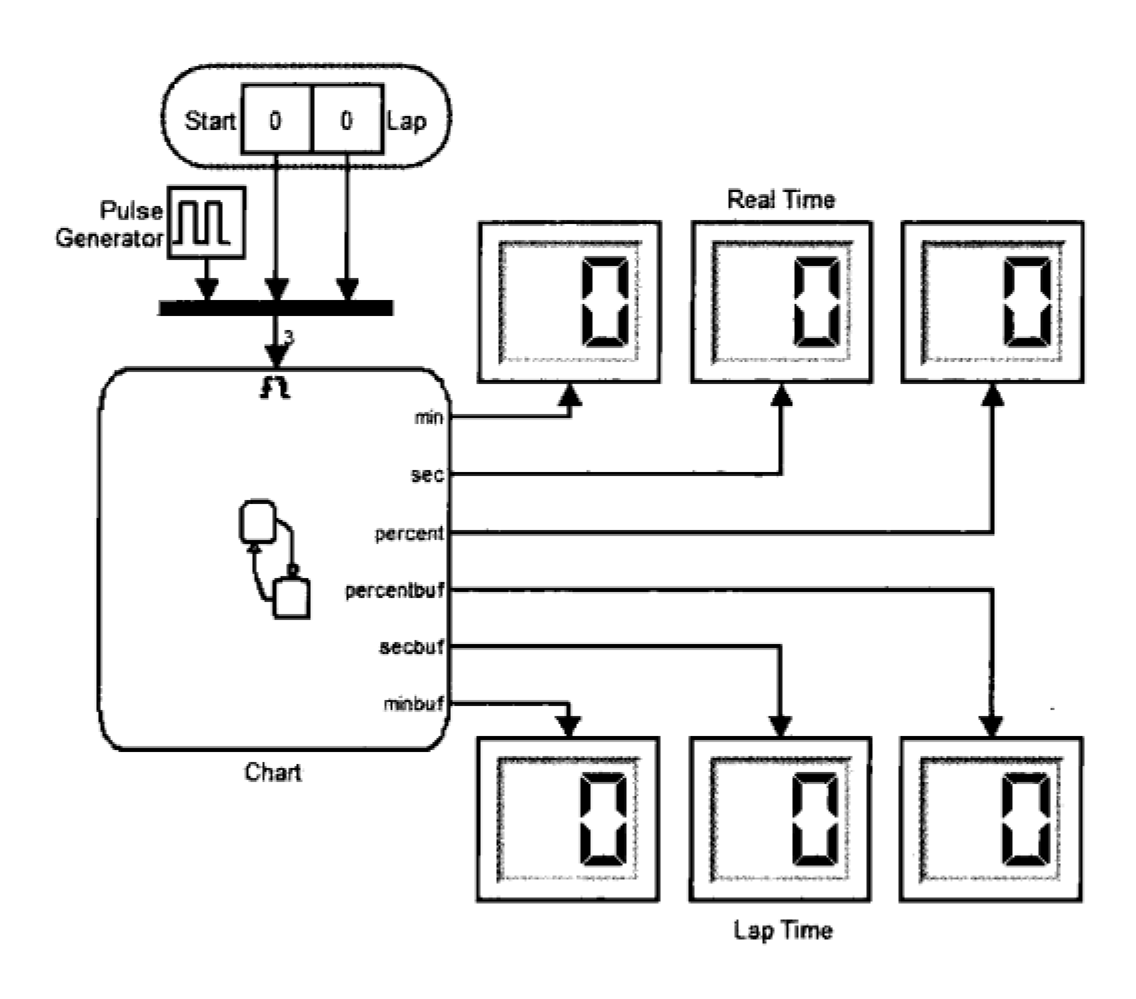


图 3.7.12 调整功能验证模型

(2) 选择 MATLAB 界面的菜单项 File→New→GUI，在打开的对话框中选择 GUI 模板为 Blank GUI，勾选下方的 Save new figure as… 复选框，指定 GUI 文件名，并确保文件路径与 Simulink 模型的一致，如图 3.7.13 所示。

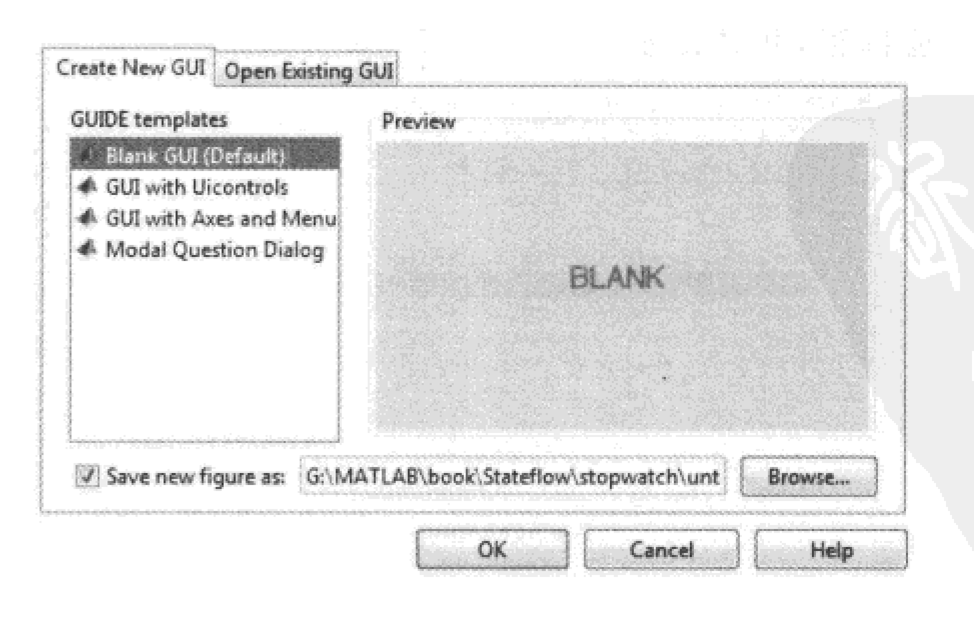


图 3.7.13 新建 GUI 界面选项

(3) 确定后,系统打开 GUI 编辑窗口与回调函数编辑窗口,如图 3.7.14 和图 3.7.15 所示。

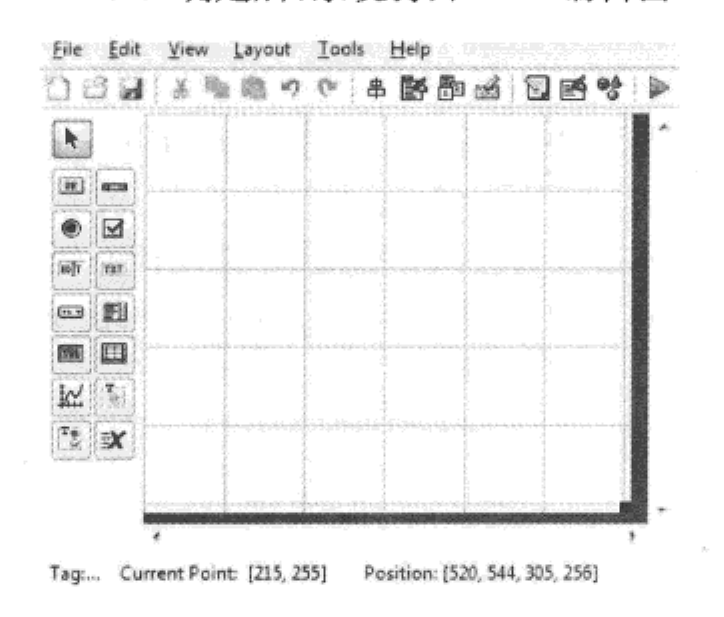


图 3.7.14 GUI 编辑窗口

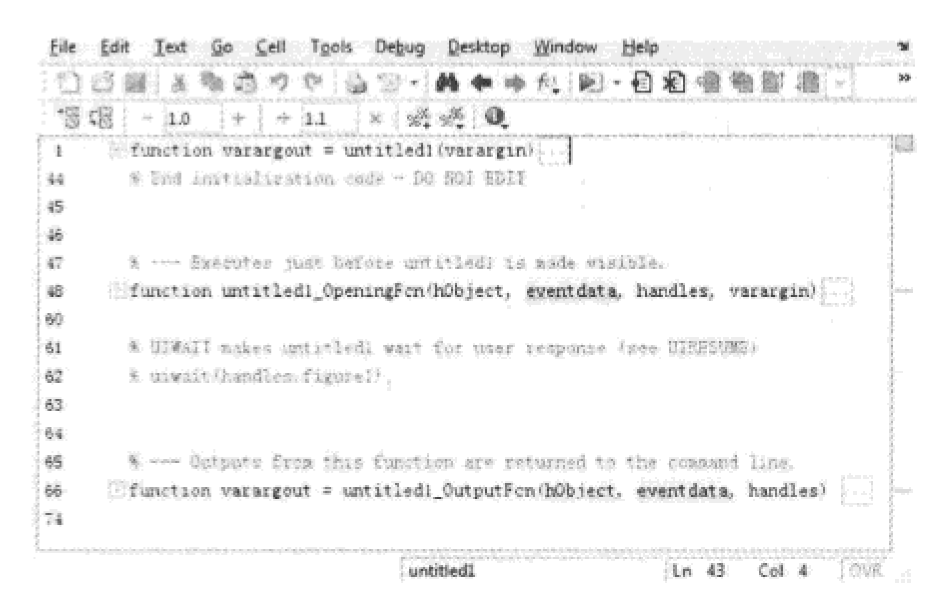


图 3.7.15 回调函数编辑窗口

(4) GUI 编辑窗口里加入两个按钮,如图 3.7.16 所示。

(5) 双击按钮,修改属性对话框 String 栏目的内容,将显示文字修改为 Start/Reset 与 LAP,如图 3.7.17 所示

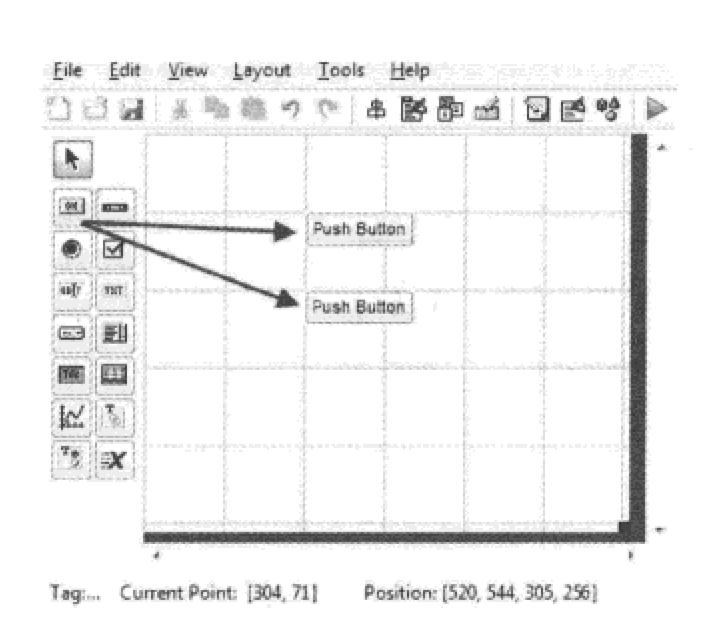


图 3.7.16 添加按钮

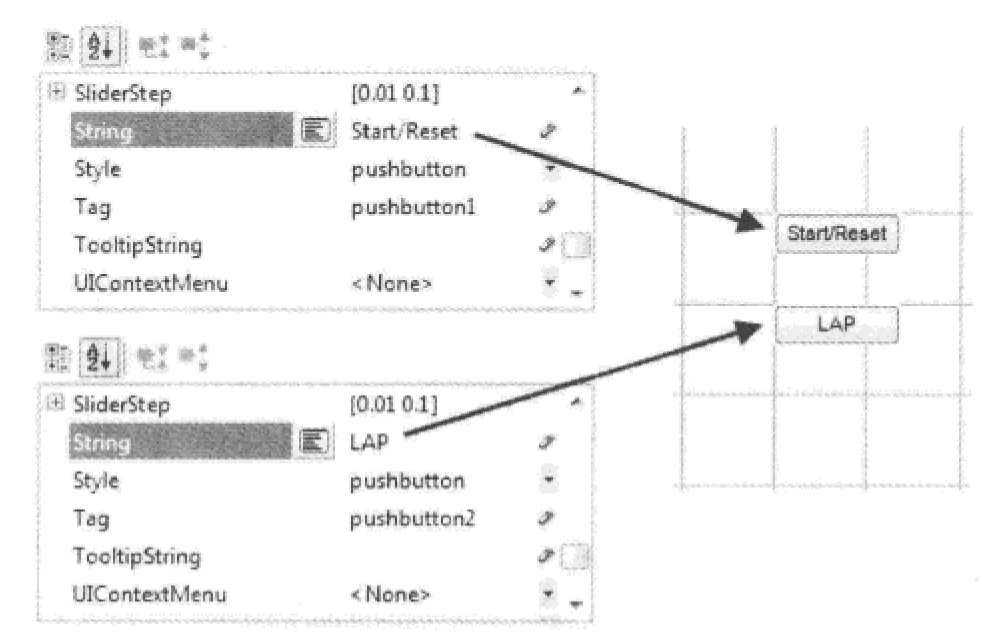


图 3.7.17 修改 String 栏目显示文字内容

(6) 选择 Start/Reset 按钮的右键菜单项 View Callbacks→Callback,系统自动定位到回调函数编辑窗口的对应位置,并高亮显示,如图 3.7.18 所示。

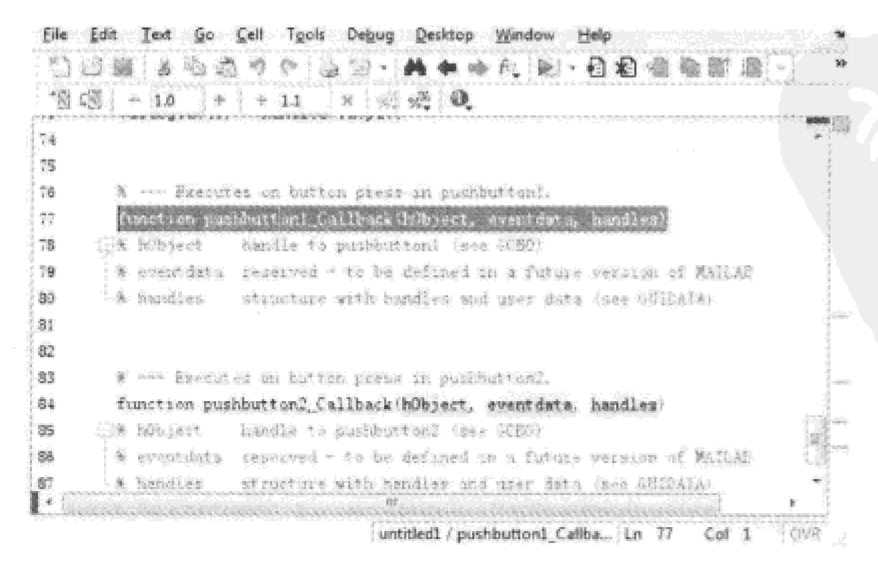
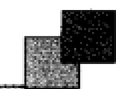


图 3.7.18 编辑回调函数



(7) 添加 Start/Reset 按钮的回调方法代码如下，其中 stopwatch_state 表示模型名称，用户应按实际调整。

```
function pushbutton1_Callback(hObject, eventdata, handles)
......
start = str2num(get_param('stopwatch_state/Start','Value'));
if start == 0
   start = 1;
else
   start = 0;
end
set_param('stopwatch_state/Start','Value', num2str(start));
```

同样的方法，添加 LAP 按钮的回调方法代码如下：

```
function pushbutton2_Callback(hObject, eventdata, handles)
......
Lap = str2num(get_param('stopwatch_state/Lap','Value'));
if Lap == 0
   Lap = 1;
else
   Lap = 0;
end
set_param('stopwatch_state/Lap','Value', num2str(Lap));
```

(8) 单击 GUI 界面编辑器的工具栏按钮▶，执行该 GUI，如图 3.7.19 所示。

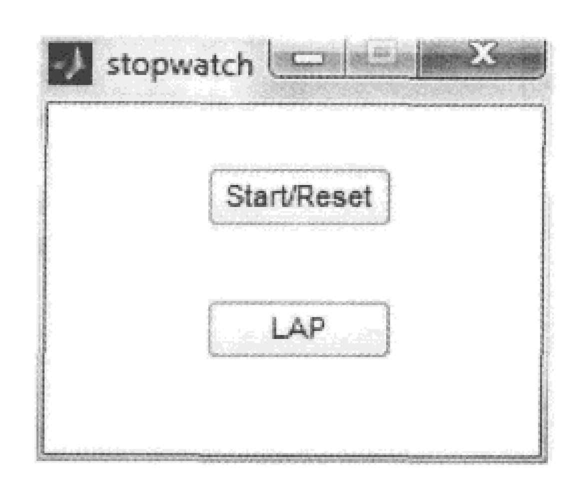


图 3.7.19 执行 GUI

这时 Start/Reset 与 LAP 按钮即可实现原先的开关功能，即使不运行模型，单击这两个按钮，模型窗口的 Start 与 Lap 常数模块的数值已能够实时变化。

(9) 选择 Simulink 模型窗口的菜单项 File→Model Properties，在 Callbacks 选项卡的 PostLoadFcn 条目，添加文字 stopwatch，如图 3.7.20 所示。这样重新打开模型窗口时，GUI 界面即可自动打开。

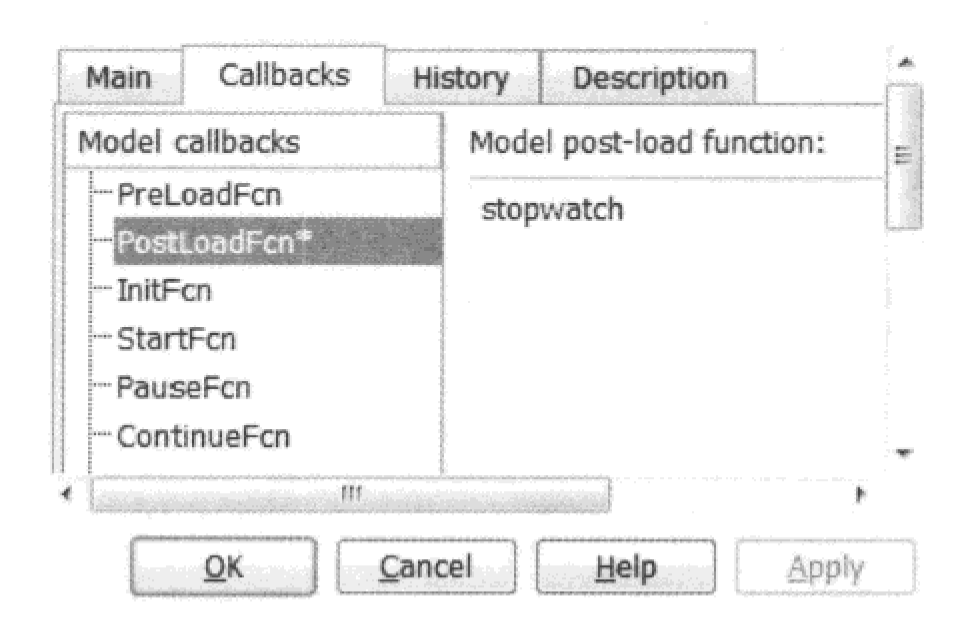


图 3.7.20　Callbacks 选项卡

3.7.2　交通灯

1. Stateflow 状态图

一个小型路口的交通灯亮灭过程可以用以下文字简单表示：南北向禁行时红灯亮起并保持 50 s，同时东西向通行，绿灯亮起，但只保持 45 s，45 s 时东西向缓行，黄灯亮起，保持 5 s；之后东西向禁行，红灯亮起并保持 50 s，同时南北向通行，绿灯亮起，同样只保持 45 s，45 s 时南北向缓行，黄灯亮起，保持 5 s，如此循环反复。

用户根据上述各节的介绍，应能快速地建立图 3.7.21 所示的简单交通灯状态图。

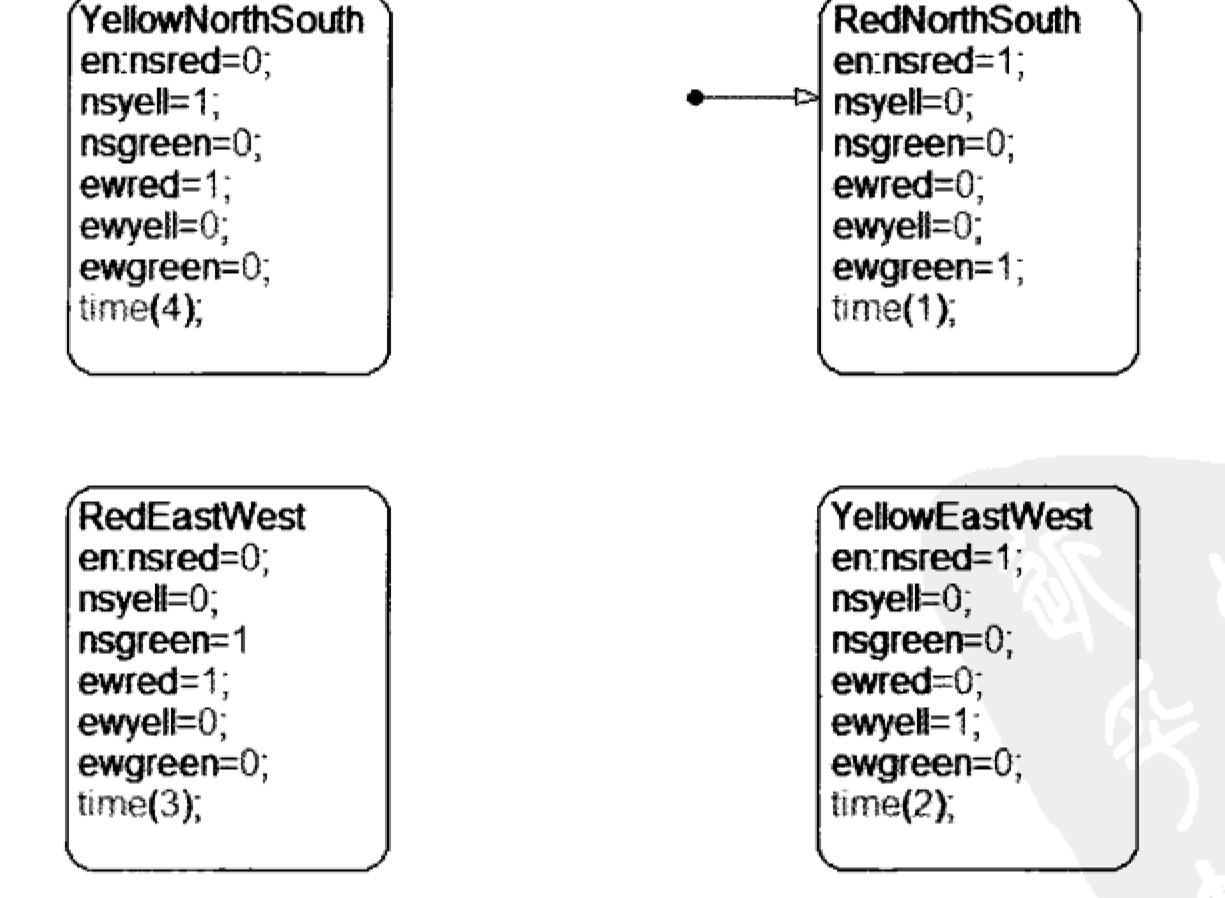
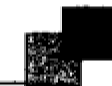


图 3.7.21　简单交通灯状态图



图中 4 个状态 RedNorthSouth、YellowEastWest、RedEastWest、YellowNorthSouth 分别代表南北向禁行、东西向缓行、东西向禁行、南北向缓行。变量 nsred、nsyell、nsgreen、ewred、ewyell、ewgreen 分别代表南北向与东西向的红灯、黄灯、绿灯，在不同的状态中，它们的亮灭以 1 或 0 表示。

简便起见，可以设置任意一个状态作为默认迁移状态，进入该状态时，便开始倒数计时，显然处于不同的状态，每个方向的倒数计时起始值是不一样的：例如红灯 50 s、黄灯 5 s、绿灯45 s。在 Stateflow 状态图中，添加一个 Embedded MATLAB 函数 time(flag)实现该功能，如图 3.7.22 所示，代码如下：

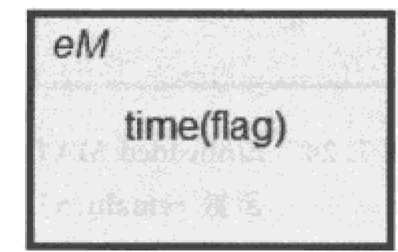


图 3.7.22 Embedded MATLAB 函数 time(flag)

```
function time(flag)
timeout = 0;                          % 计时终了标志清零
switch flag                           % 判断当前所处的通行状态
case 1                            % 南北向禁行时
     nssec = red;                     % 取变量 red 作为南北向计时起始值
     ewsec = green;                   % 取变量 green 作为东西向计时起始值
     sec = green * 10;                % 提高计时精度,放大倒数计时变量
case 2                            % 东西向缓行时
     ewsec = red-green;               % 取变量 red-green 作为东西向计时起始值
     sec = (red-green) * 10;
case 3                            % 东西向禁行时
     ewsec = red;                     % 取变量 red 作为东西向计时起始值
     nssec = green;                   % 取变量 green 作为南北向计时起始值
     sec = green * 10;
case 4                            % 南北向缓行时
     nssec = red-green;               % 取变量 red-green 作为南北向计时起始值
     sec = (red-green) * 10;
end
```

读取了计时起始值，即开始倒数计时，这里沿用 3.7.1 节的计时流程图，稍作改造并封装为图形函数 count()，供各状态调用，如图 3.7.23 所示。每当事件 TIC 发生，倒数计时变量 sec 减 1，当 sec=0 时，计时终了标志 timeout 置 1，状态应发生迁移。

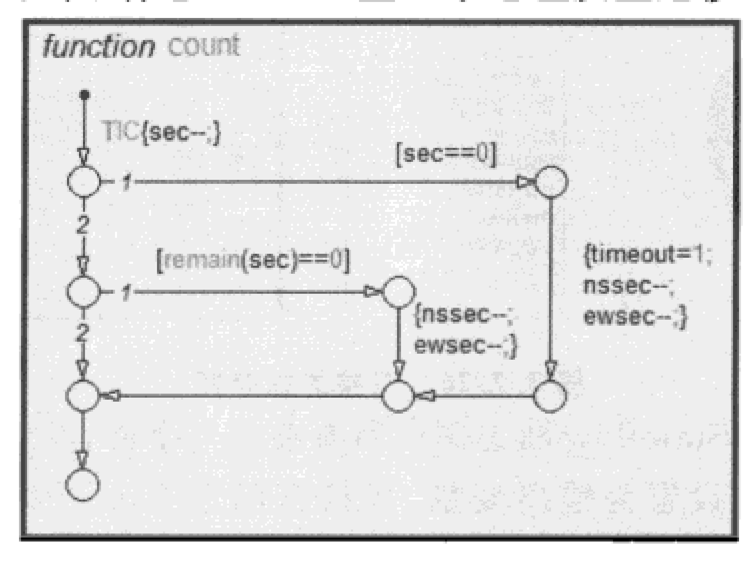
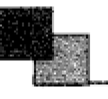


图 3.7.23 图形函数 count()



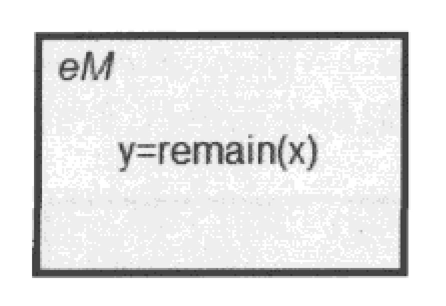


图 3.7.24 Embedded MATLAB 函数 remain(x)

为了提高计时精度，函数 time(flag)曾将计时变量 sec 放大了 10 倍，只有当计时变量的个位为 0 时，才表示计得 1 秒，南北向、东西向的计时值才可以减 1，因此需要用到取余函数 rem()。不过在流程图的迁移条件里不能直接使用该函数，为此将其封装成 Embedded MATLAB 函数 remain(x)，如图 3.7.24 所示，代码如下：

```
function y = remain(x)
y = rem(x,10);
```

状态迁移过程如图 3.7.25 所示。

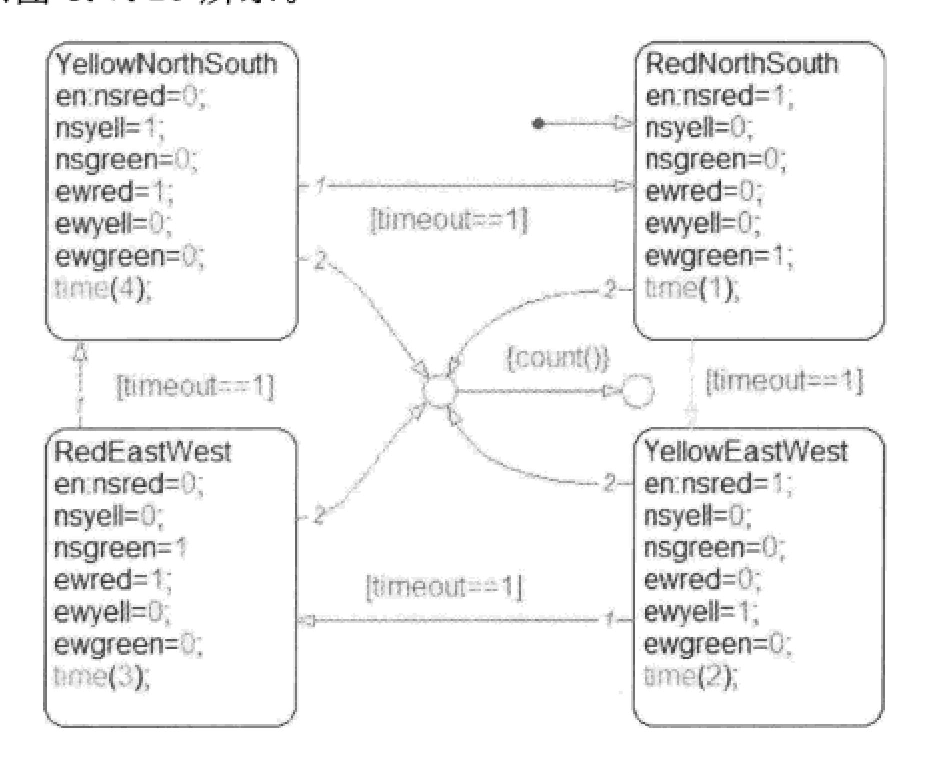


图 3.7.25 状态迁移过程

完整的状态图如图 3.7.26 所示。

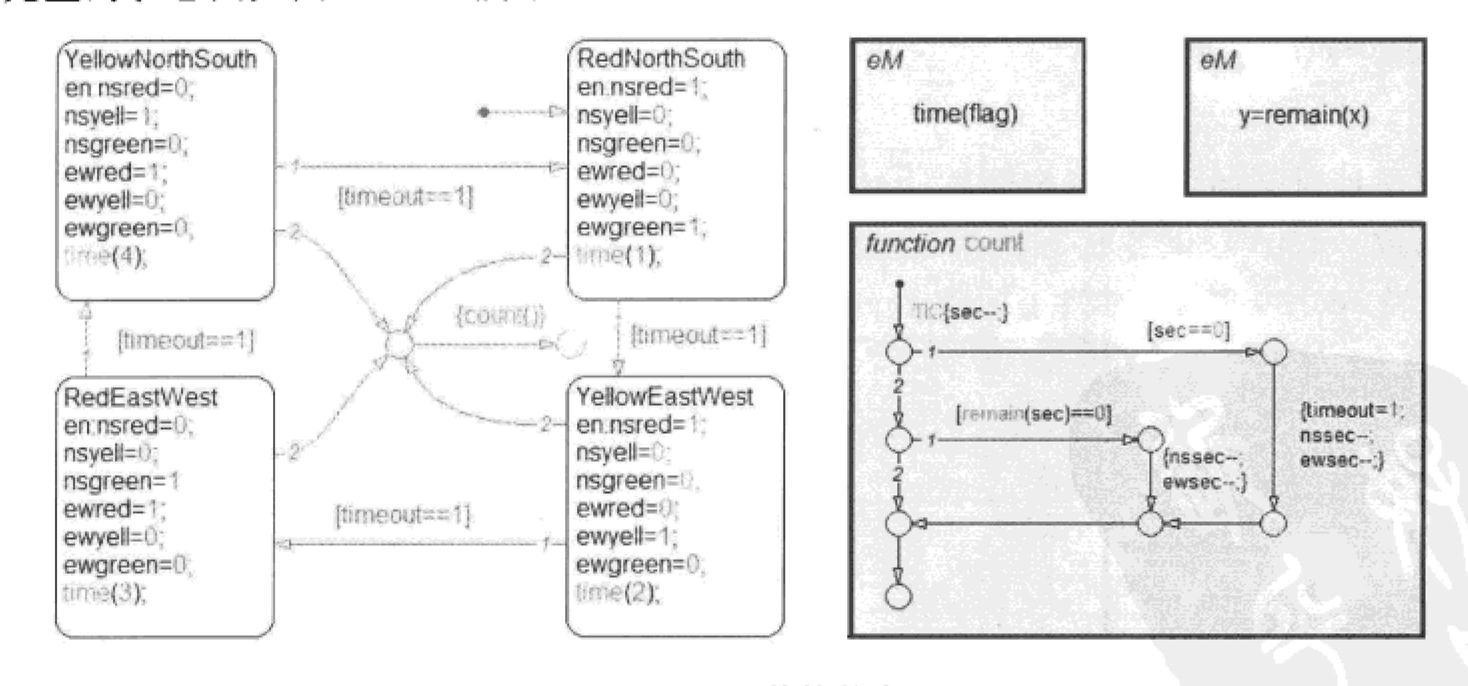
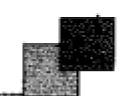


图 3.7.26 完整的状态图

图 3.7.27 列出了整个 Stateflow 状态图的所有变量与事件，南北向、东西向的三盏指示灯以及计时值 nssec、ewsec 均设置为输出变量，而状态标志 flag、计时变量 sec、计时终了标志 timeout 只需作为内部变量，TIC 作为时钟信号触发类型选为上升沿。



Name	Scope	Port	DataType	Trigger	Size	InitialValue	CompiledSize	Compiled
ewred	Output	1	double					unknown
ewyell	Output	2	double					unknown
ewgreen	Output	3	double					unknown
nsred	Output	4	double					unknown
nsyell	Output	5	double					unknown
nsgreen	Output	6	double					unknown
nssec	Output	7	double					unknown
ewsec	Output	8	double					unknown
flag	Local		double					unknown
sec	Local		double					unknown
timeout	Local		double					unknown
red	Input	1	double					unknown
green	Input	2	double					unknown
TIC	Input	1		Rising				

图 3.7.27　变量与事件列表

2. Simulink 建模

完成 Stateflow 状态图之后，在 Simulink 模块库中找到图 3.7.28～图 3.7.31 所示的模块，添加到模型。

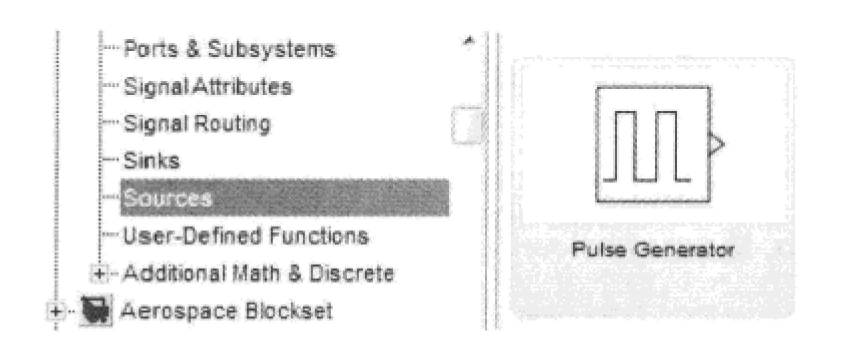
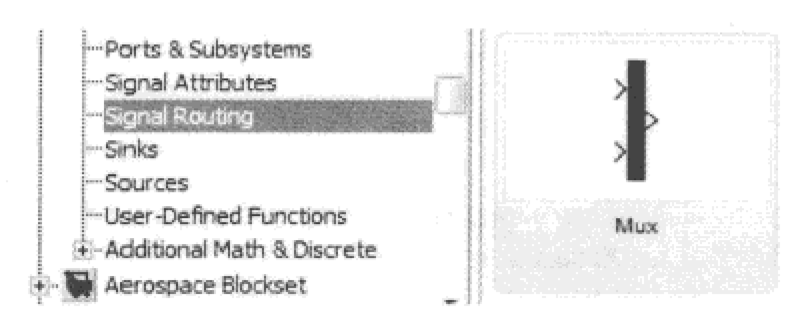


图 3.7.28　脉冲发生器模块

图 3.7.29　常数模块



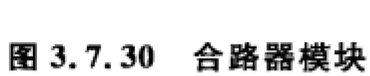


图 3.7.30　合路器模块

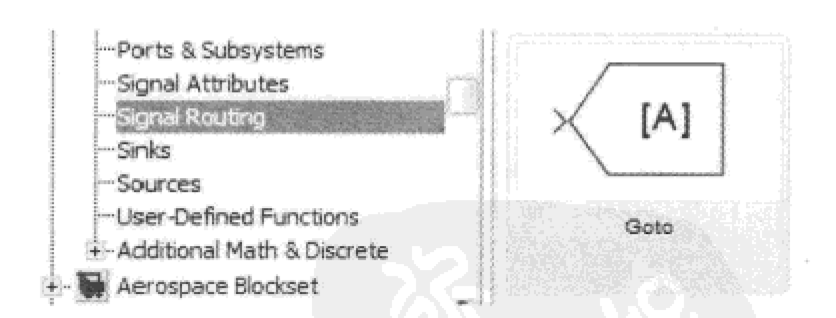
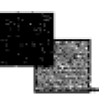


图 3.7.31　Goto 模块

按图 3.7.32 连接，任意设置常数 red 与 green 的值，例如 10、7，这代表红灯与绿灯的点亮时长。

继续在模型中添加图 3.7.33～图 3.7.36 所示模块。



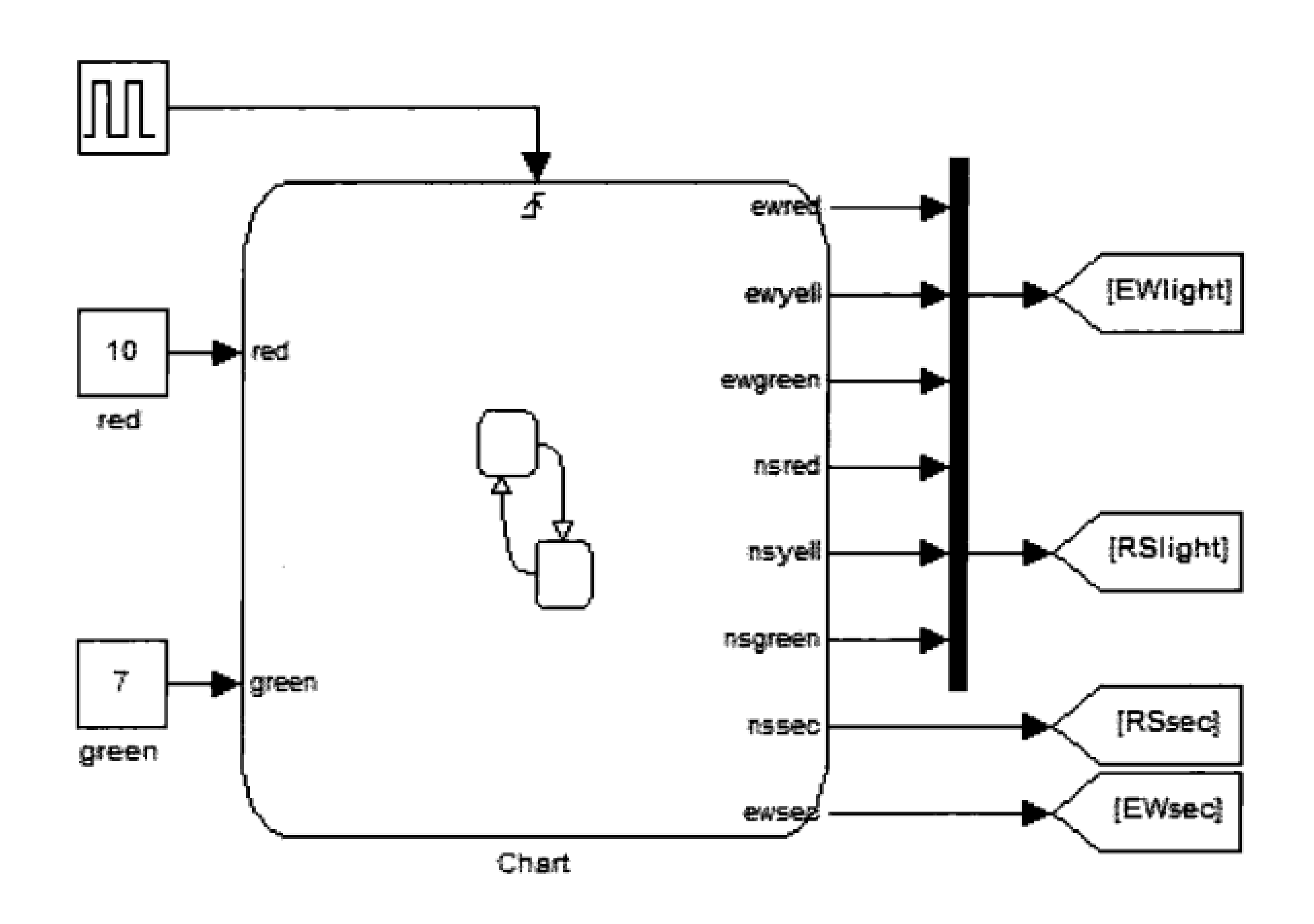


图 3.7.32 Stateflow 部分模块连接

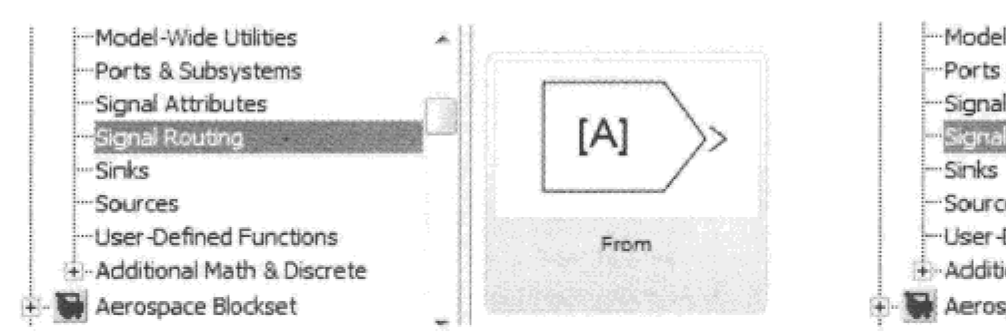


图 3.7.33 From 模块

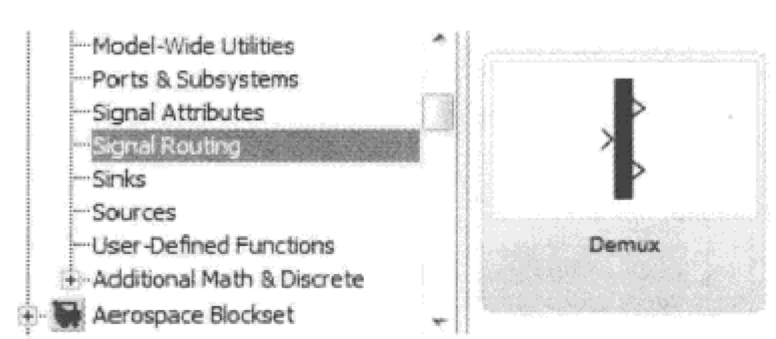


图 3.7.34 分路器模块

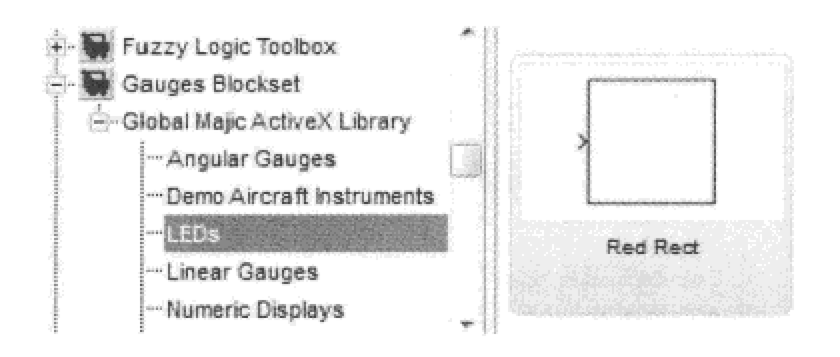


图 3.7.35 红色方块模块

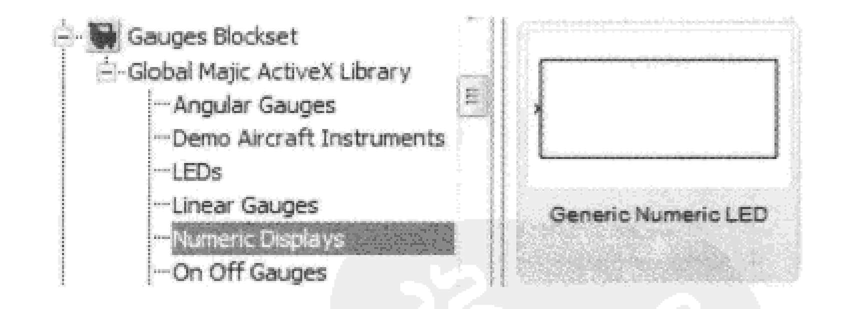


图 3.7.36 数字 LED 模块

该状态图的连线过多，如果仍使用连线直接相连，必导致图形杂乱无章，信号流向难以分辨。使用 From、Goto、Demux 模块，可以大大地简化图形，如图 3.7.37 所示。

Red Rect 模块若不作任何修改，它只能显示红色，用户可以根据需要设置模块的背景色、外框、亮灯颜色、灭灯颜色等，如图 3.7.38、图 3.7.39 所示。



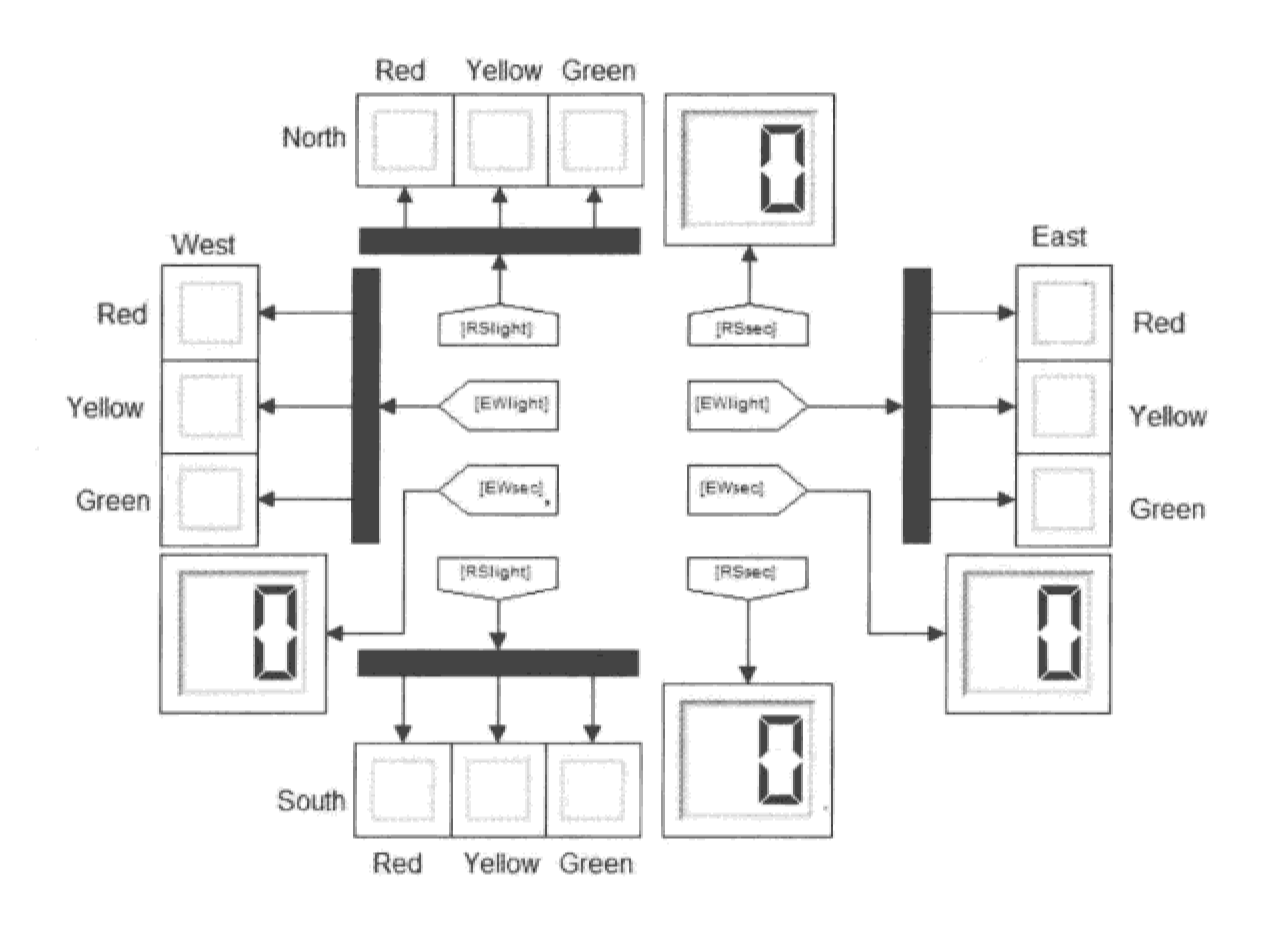


图 3.7.37 显示部分模块连接

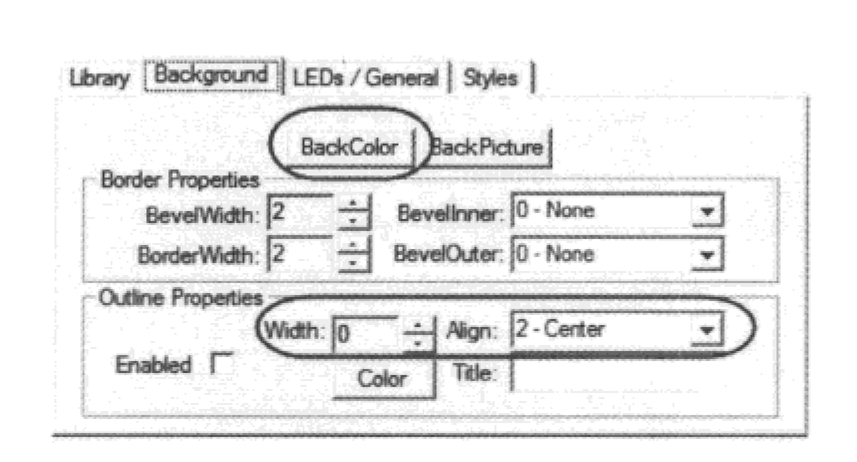


图 3.7.38 修改红色方块模块的背景色

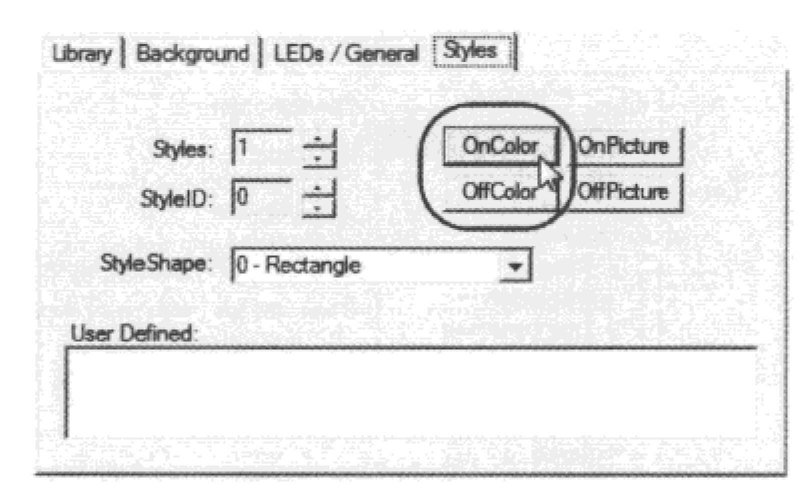


图 3.7.39 修改红色方块模块的亮灭色

南北、东西两向的倒数计时最多为两位十进制数，因此设置数码管其显示位数为 2 位整数、0 位小数，如图 3.7.40 所示。

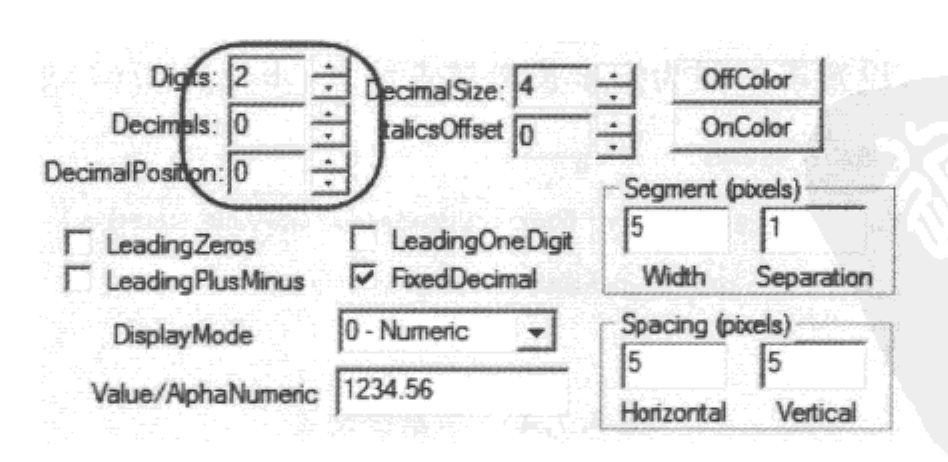


图 3.7.40 修改数字 LED 模块的显示位数

3. 模型设置及仿真

选择模型主窗口的菜单项 Simulation→Configuration Parameters…，如图 3.7.41 所示，打开模型参数对话框，如图 3.7.42 所示。

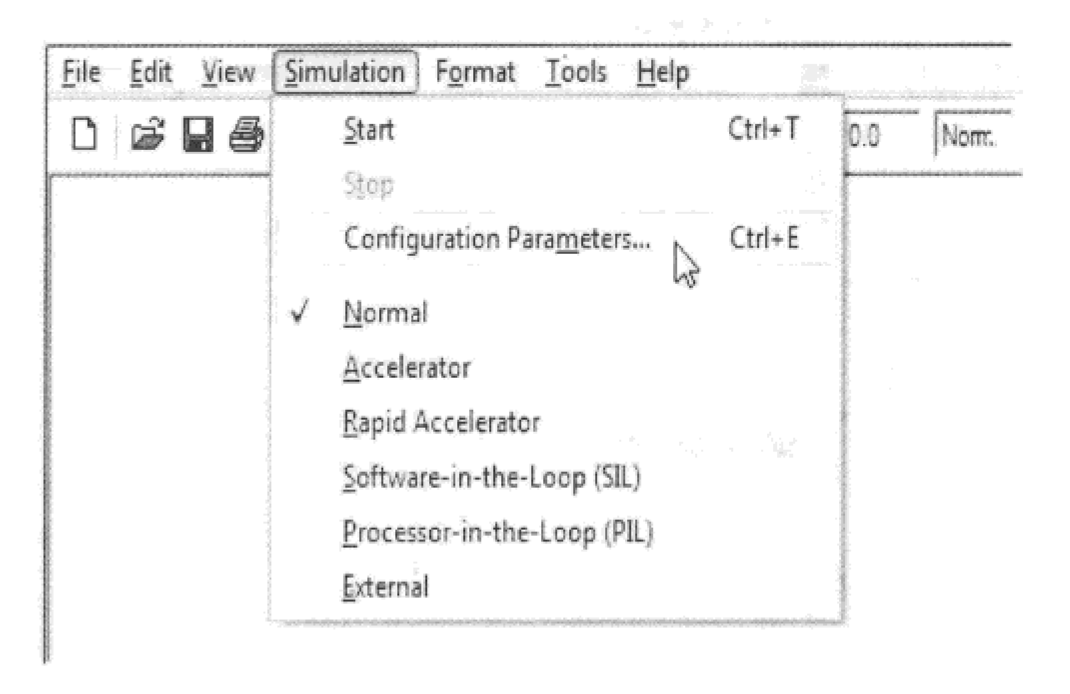


图 3.7.41 模型参数设置菜单

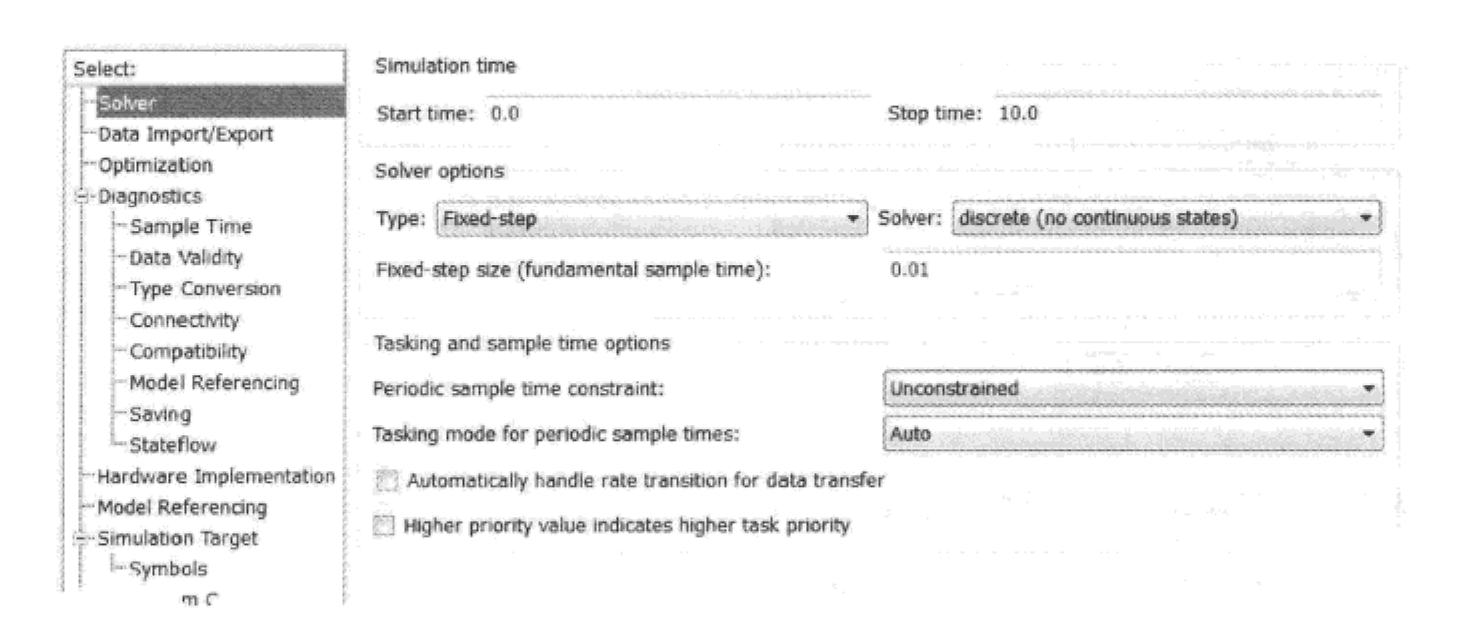


图 3.7.42 参数设置对话框

在 Solver 界面中，设置求解器为定步长离散求解器，步长为 0.01，如图 3.7.43 所示。

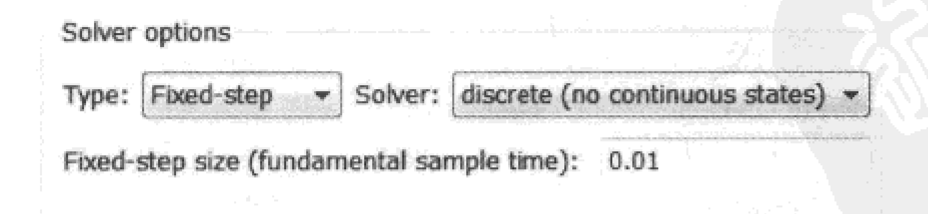
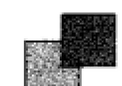


图 3.7.43 求解器设置

双击脉冲发生器模块，根据实际需要设置脉冲周期与脉冲宽度，如图 3.7.44 所示。

完成以上设置后执行仿真，模型即按设计开始运行，如图 3.7.45～图 3.7.48 所示。



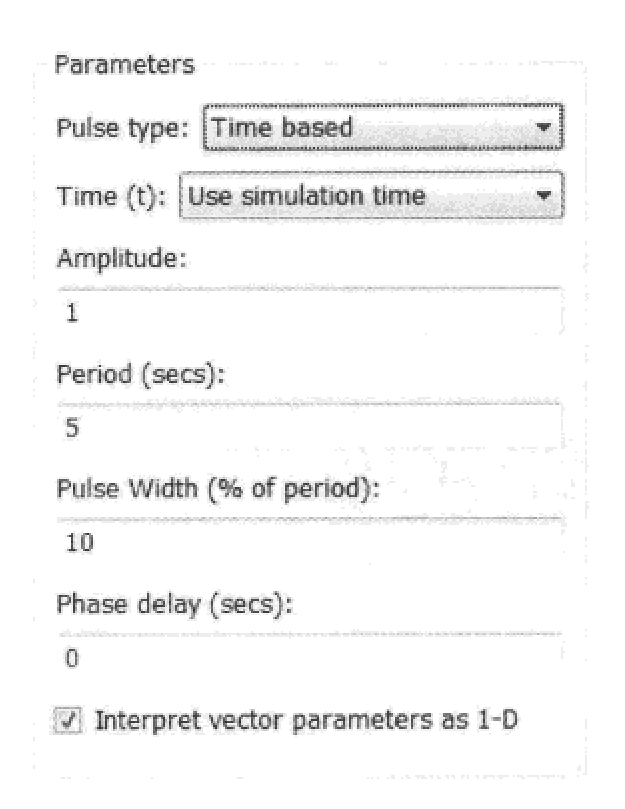


图 3.7.44　脉冲发生器参数

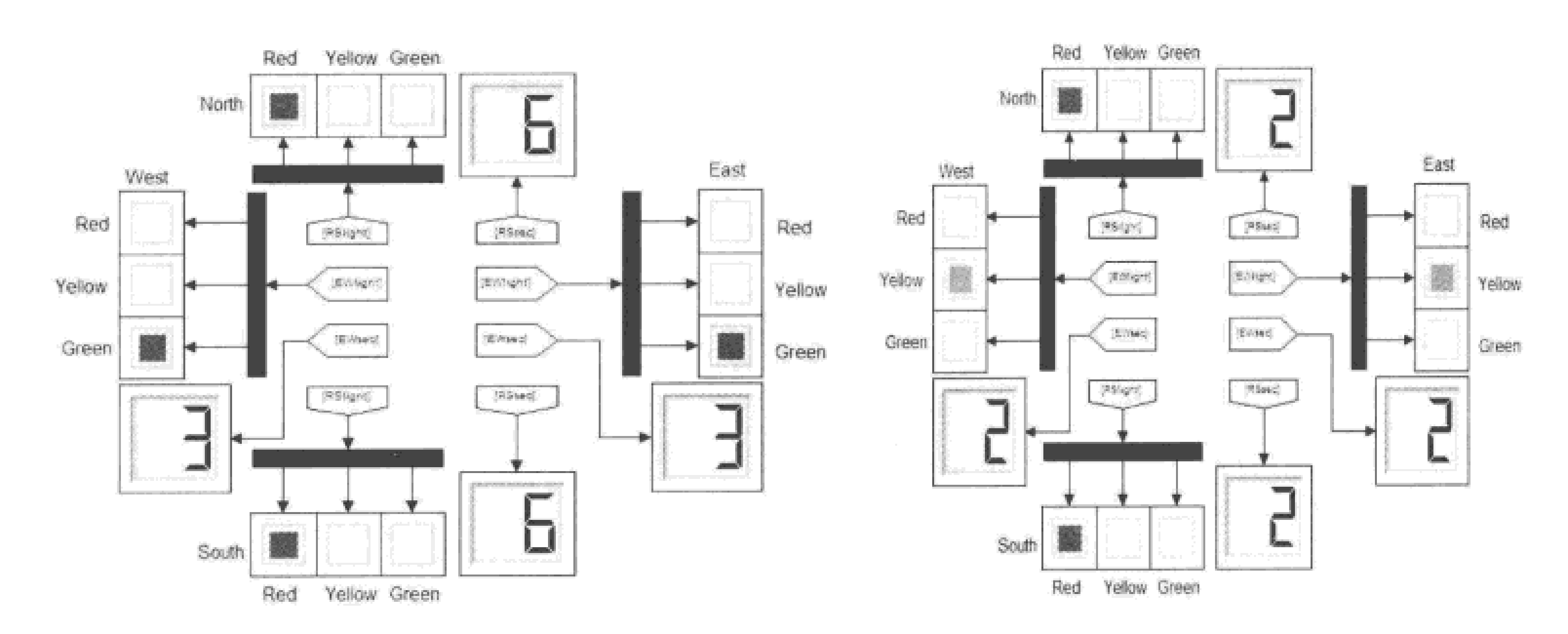


图 3.7.45　南北向禁行

图 3.7.46　东西向缓行

图 3.7.47　东西向禁行

图 3.7.48　南北向缓行

第 4 章

设备驱动模块的编写

基于模型设计的核心是系统模型，如图 4.0.1 所示，而模型又是由一个个的模块组成，其他部分如验证与测试、设计仿真等只是保证该系统模型能正常工作的手段。尽管 MathWorks 公司为 MATLAB 用户提供了一千多个预定义模块和一些算法，同时诸多芯片厂家也为各的芯片开发了设备驱动模块，但仅仅依靠这些现有模块还不能解决所有的实际问题。即使能用这些基本模块组成应用系统，但该系统结构复杂，存在改一点而动全身的问题，给系统升级带来麻烦；同时，系统的运算速度也因存在大量的中间数据交换而受到限制。用户创建自己的设备驱动模块，是基于模型设计必须掌握的基本技能，也是难点和核心内容之一。

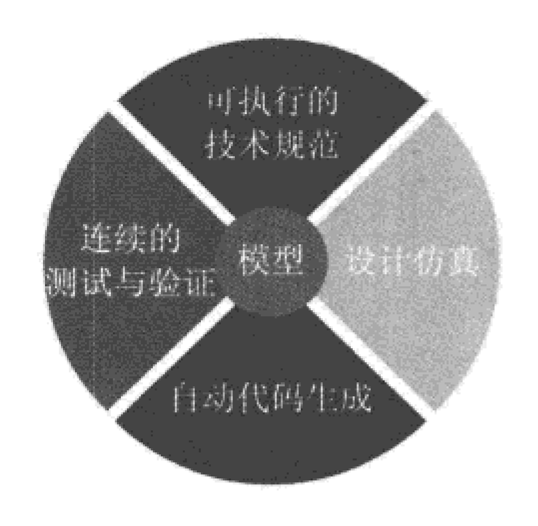
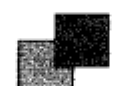


图 4.0.1　基于模型设计的核心

创建设备驱动模块的方法一般是使用 C MEX S-function 和 Embedded MATLAB。Embedded MATLAB 比较容易上手(已在第 1 章介绍)，不过它目前支持的函数还不太多，应用受到一定的限制。本章将重点介绍用 C MEX S-function 创建用户自定义设备驱动模块的方法，包括 noninlined S-function 和 Inlined S-function 模块。noninlined S-function 模块主要用于系统仿真(RTW 也可为其生成 C 代码)，而 Inlined S-function 模块则为模块产生嵌入式实时 C 代码。

本章主要内容如下：

- 建立简单的 S-Function 模块。
- S-Function 与 S-Function Builder 原理。
- 应用。



4.1　创建S函数模块的示例

本章主要介绍如何创建用户设备驱动模块，而编写C MEX S函数是一项很有挑战性的工作。为了帮助读者在现有的知识水平下，尽快熟悉设备驱动模块的编写方法，我们摈弃了传统的从理论指导实践的方法，采用自然的认知过程——从感性认识到理性认识。本节利用一个简单的例子——算术乘法，向读者展示S函数的各种编写方法，同时让读者产生一些疑问和思考，针对这些问题，在后续小节逐一解答。

现有的算法大多数已用C语言实现了，并得到了广泛的应用，对于这些经过验证的C代码，用户没有必要重新编写。因此合理正确地利用现有的C代码，可加快S函数的编写。

集成C代码编写S函数的方法有以下3种：

(1) 手工编写 Wrapper S函数：这是最直接且最灵活的方法，但这需要用户详细了解S函数的结构与S函数的API函数(回调方法)。如果要生成内联S函数的代码，还需要TLC (Target Language Compiler)文件，编写过程难度较大。

(2) 利用S-Function Builder生成S函数：这是最简单且最直观的一种方法，用户可以不需事先了解S函数的结构与API函数，当然如果用户具备这些知识，使用起来更会得心应手。它同时可以生成TLC文件，用以生成内联S函数的代码。不过S-Function Builder调用的API函数只是S函数API的一个子集，因此生成的S函数，功能受到一定限制。

(3) 使用代码继承工具(Legacy Code Tool，LCT)创建S函数模块：代码继承工具事实上是一组MATLAB命令，命令的参数对应着S函数的各个字段，因此要求用户应事先了解S函数的基本结构。与S-Function Builder一样，LCT工具也可以生成TLC文件，但它所调用的API函数较S-Function Builder来得更少。

本节以现有的C代码doubleIt.c为例，说明整合现有C代码，创建S-function模块的3种方法：

```
double doubleIt(double u) //doubleIt.c
{
    return(u * 2.0);
}
```

4.1.1　手工编写 Wrapper S函数

由于手工编写S-function需要事先了解它的结构，因此本节仅简要说明代码意义，4.2节将详细介绍S-function的结构。Wrapper S函数的原理如图4.1.1所示。

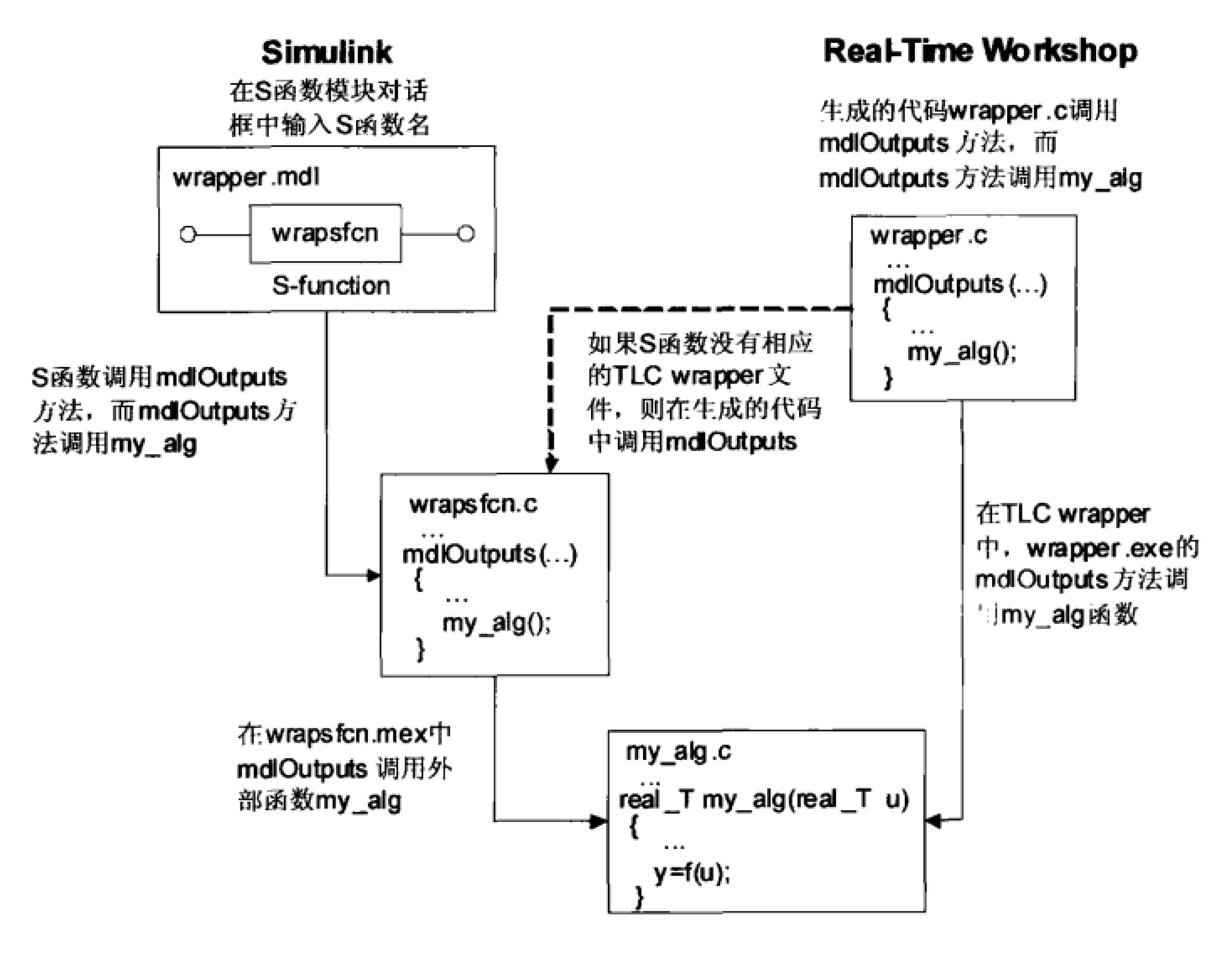


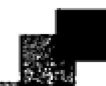
图 4.1.1　Wrapper S 函数的原理

1. 手写 C MEX S 函数代码

若读者暂时不能理解代码中的注释，可留待后期了解了相关内容再回来阅读。

```
#define S_FUNCTION_NAME wrapsfcn        // S 函数名
#define S_FUNCTION_LEVEL 2              // S 函数的级别为 2
#include "simstruc.h"
extern real_T doubleIt(real_T u);       // 声明外部函数
```

```
// 模型初始化
static void mdlInitializeSizes(SimStruct *S){
    ssSetNumSFcnParams(S, 0);          // S 函数参数个数为 0
    ssSetNumContStates(S, 0);          // 连续状态个数为 0
    ssSetNumDiscStates(S, 0);          // 离散状态个数为 0
    // 如果输入口个数不为 1,则返回
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);              // 第一个输入口维度
    ssSetInputPortDirectFeedThrough(S, 0, 1);// 第一个输入口为直馈
    // 如果输出口个数不为 1,则返回
    if (!ssSetNumOutputPorts(S,1)) return;
    ssSetOutputPortWidth(S, 0, 1);       // 第一个输出口维度
    ssSetNumSampleTimes(S, 1);           // 采样率个数为 1
```



```
}
```

```
// 采样时间初始化
static void mdlInitializeSampleTimes(SimStruct * S){
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME);// 采样时间为继承
    ssSetOffsetTime(S, 0, 0);           // 采样时间偏移量为 0
}
```

```
// 模型输出
static void mdlOutputs(SimStruct * S, int tid){
    InputRealPtrsType uPtrs  =  ssGetInputPortRealSignalPtrs(S,0);
    real_T * y  =  ssGetOutputPortRealSignal(S,0);
     * y  =  doubleIt( * uPtrs[0]);        // 调用外部函数
}
```

```
// 模型结束
static void mdlTerminate(SimStruct * S){
    UNUSED_ARG(S);
}
```

```
# ifdef MATLAB_MEX_FILE                // 此 5 行代码,一般不应删除
# include "simulink.c"
# else
# include "cg_sfun.h"
# endif
```

将 doubleIt.c 保存在 wrapsfcn.c 同一目录下,在命令行输入以下代码,编译该 S 函数。

```
mex wrapsfcn.c doubleIt.c
```

2. 功能验证模型

新建一个 Simulink 模型,并加入一个 S 函数模块,如图 4.1.2 所示。

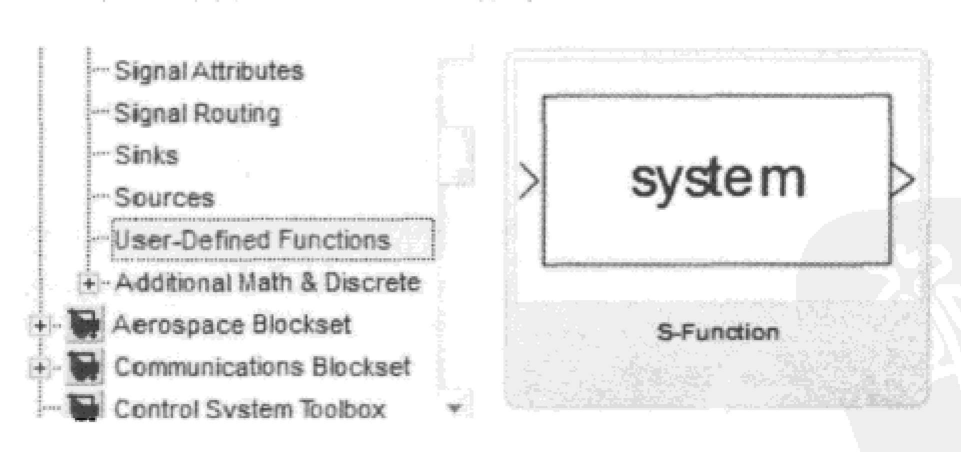


图 4.1.2 S函数模块

设置其对应的 S 函数名为 wrapsfcn,如图 4.1.3 所示。注意,该函数名不包括任何扩展名。

在该模型里继续添加信号源、增益、示波器等模块,如图 4.1.4 所示。

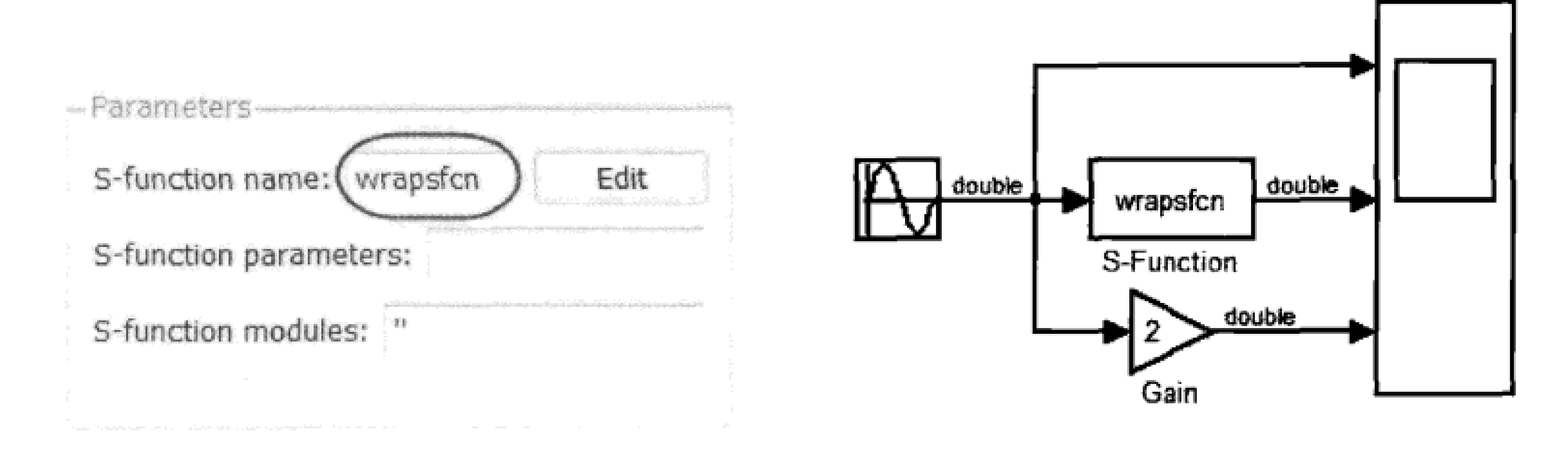


图 4.1.3 设置 S 函数名

图 4.1.4 功能验证模型

单击"仿真"按钮，示波器显示 S 函数输出结果与增益模块的一致，说明 S 函数实现了设计的功能，如图 4.1.5 所示。

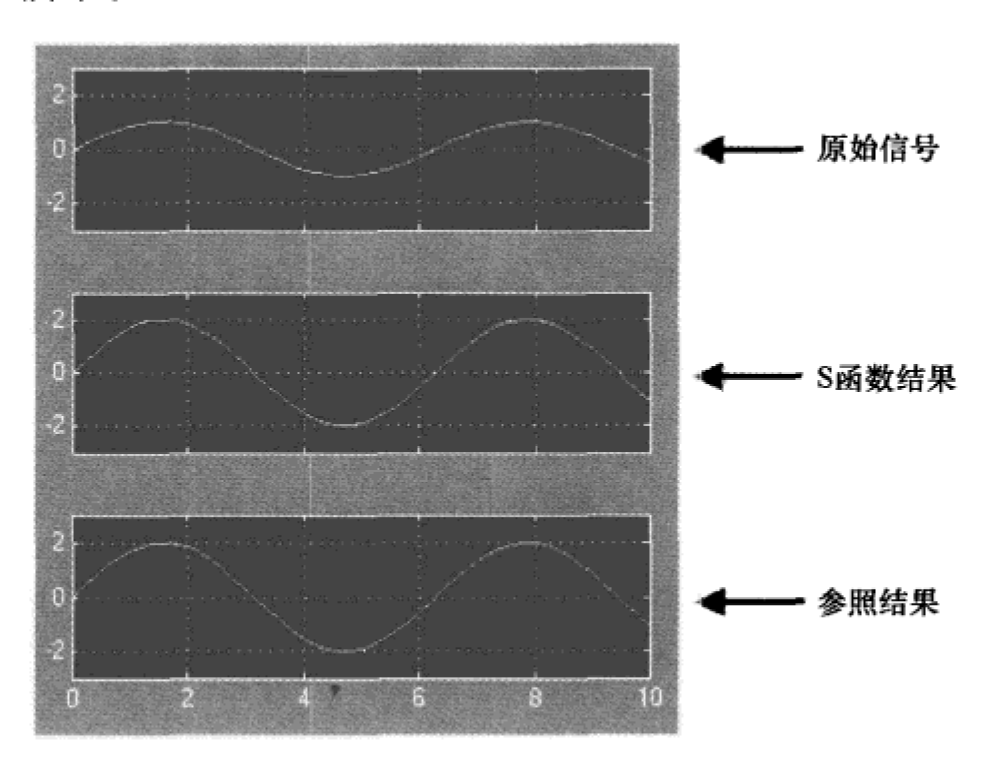


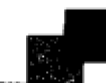
图 4.1.5 输出结果

3. TLC 文件

为了能够使用 Real-Time Workshop(RTW)代码生成工具，定制系统模型的实时 C 代码生成过程，必须创建内联的 S 函数，这时还需要手写 TLC 文件。

以下是 wrapsfcn.c 对应的 TLC 文件 wrapsfcn.tlc，其中 BlockTypeSetup 函数为 doubleIt.c 声明了一个函数原型，Outputs 函数则告诉 RTW 代码生成工具如何将 doubleIt.c 函数作为内联函数调用。

```
% implements "wrapsfcn" "C"
% % Function: BlockTypeSetup
% % Abstract:
% %        Create function prototype in model.h as:
% %        "extern double doubleIt(double u);"
% function BlockTypeSetup(block, system) void
```



```
  %openfile buffer
  %% ASSIGNMENT: PROVIDE ONE LINE OF CODE AS A FUNCTION PROTOTYPE
  extern double doubleIt(double u);
  %closefile buffer
  %<LibCacheFunctionPrototype(buffer)>
%endfunction
```

```
%% Function: Outputs
%%        y = doubleIt( u );
%function Outputs(block, system) Output
  /* %<Type> Block: %<Name> */
  %assign u = LibBlockInputSignal(0, "", "", 0)
  %assign y = LibBlockOutputSignal(0, "", "", 0)
  %% PROVIDE THE CALLING STATEMENT FOR "doubleIt"
  %<y> = doubleIt( %<u> );
%endfunction
```

4.1.2　代码继承工具(Legacy Code Tool)

代码继承工具，可以较快地将现有 C 代码生成一个 S 函数模块，它的工作原理如图 4.1.6 所示。

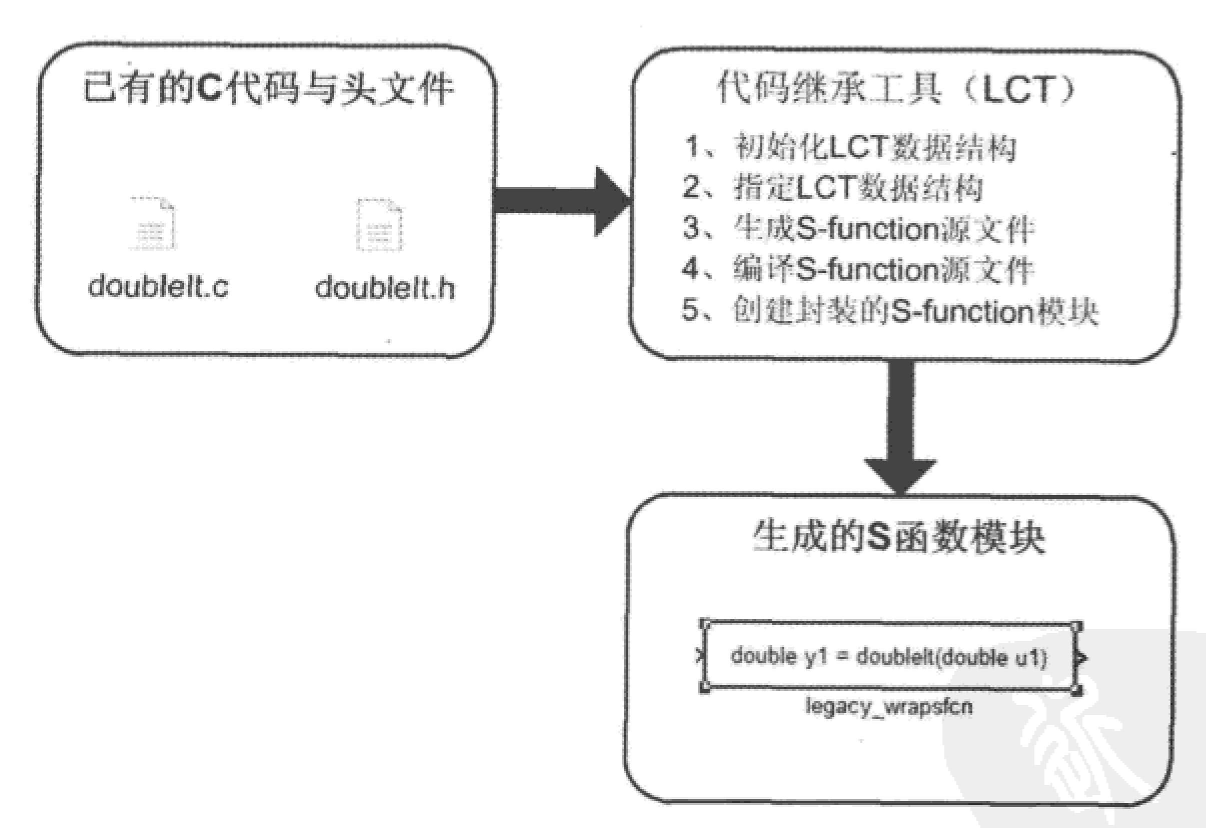


图 4.1.6　代码继承工作原理

1. LCT 脚本

将 doubleIt.c 与 doubleIt.h 放在同一目录下，运行以下脚本，系统将创建一个带有 S 函数模块的 Simulink 模型，并编译一个名为 legacy_wrapsfcn.c 的 S 函数，再生成对应的 TLC 文件 legacy_wrapsfcn.tlc。

```
def = legacy_code('initialize');                    % 创建 LCT 数据结构
def.SourceFiles = {'doubleIt.c'};                   % 指定 C 源代码
def.HeaderFiles = {'doubleIt.h'};                   % 指定 C 头文件
def.SFunctionName = 'legacy_wrapsfcn';              % 指定 S 函数名
def.OutputFcnSpec = 'double y1 = doubleIt(double u1)'; % 指定输出函数
def.SampleTime = [-1,0];                            % 指定采样时间为继承
legacy_code('slblock_generate', def);               % 生成 S 函数模块
legacy_code('sfcn_cmex_generate', def);             % 生成 C MEX S 函数
legacy_code('compile', def);                        % 编译 C MEX S 函数
legacy_code('sfcn_tlc_generate', def);              % 生成 TLC 文件
```

2. S 函数与 TLC 文件

以下是 S 函数 legacy_wrapsfcn.c 的 mdlOutputs 回调方法部分,用户可对照比较 4.1.1 节手写代码的 mdlOutputs 部分。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
  real_T *u1 = (real_T *) ssGetInputPortSignal(S, 0);
  real_T *y1 = (real_T *) ssGetOutputPortSignal(S, 0);
  *y1 = doubleIt( *u1);
}
```

LCT 工具生成的 TLC 文件 legacy_wrapsfcn.tlc,与手写文件相比,同样有 BlockTypeSetup 与 Outputs 函数,不同的是,它定义了 BlockInstanceSetup 与 BlockOutputSignal 函数。

```
%implements legacy_wrapsfcn "C"
%% Function: BlockTypeSetup
%function BlockTypeSetup(block, system) void
    %% The Target Language must be C
    %if::GenCPP==1 %% 定义全局变量
      %<LibReportFatalError("This S-Function generated by the Legacy Code Tool must be only used
    with the C Target Language")>
    %endif
    %<LibAddToCommonIncludes("doubleIt.h")>
    %<LibAddToModelSources("doubleIt")>
%endfunction
```

```
%% Function: BlockInstanceSetup
%function BlockInstanceSetup(block, system) void
    %<LibBlockSetIsExpressionCompliant(block)>
%endfunction
%% Function: Outputs
%function Outputs(block, system) Output
      %if !LibBlockOutputSignalIsExpr(0)
      %assign u1_val = LibBlockInputSignal(0, "", "", 0)
```

```
        %assign y1_val = LibBlockOutputSignal(0, "", "", 0)
        %<y1_val> = doubleIt( %<u1_val>);
        %endif
    %endfunction
```

```
    %% Function: BlockOutputSignal
    %function
BlockOutputSignal(block,system,portIdx,ucv,lcv,idx,retType) void
      %assign u1_val = LibBlockInputSignal(0, "", "", 0)
      %assign y1_val = LibBlockOutputSignal(0, "", "", 0)
      %switch retType
        %case "Signal"
          %if portIdx == 0
            %return "doubleIt( %<u1_val>)"
            %else
              %assign errTxt = "Block output port index not supported:
%<portIdx>"
              %<LibBlockReportError(block,errTxt)>
            %endif
          %default
            %assign errTxt = "Unsupported return type: %<retType>"
            %<LibBlockReportError(block,errTxt)>
      %endswitch
    %endfunction
```

3. 功能验证模型

LCT 脚本运行后，系统将自动建立一个模型，其中包含生成的 S 函数模块(图 4.1.7)。

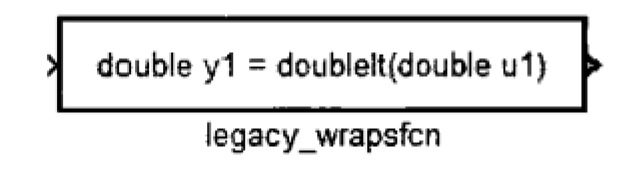


图 4.1.7　生成的 S 函数模块

在该模型里继续添加信号源、增益、示波器等模块(图 4.1.8)。

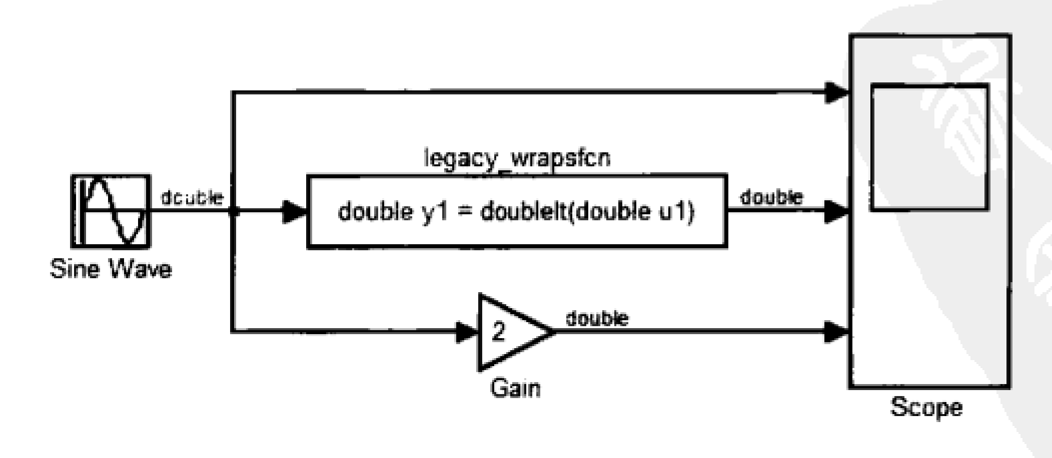


图 4.1.8　功能验证模型

在模型参数选项里选择合适的求解器，单击“仿真”按钮，得到与图 4.1.5 所示一致的仿真结果。如图 4.1.9 所示，S 函数实现了设计的功能。

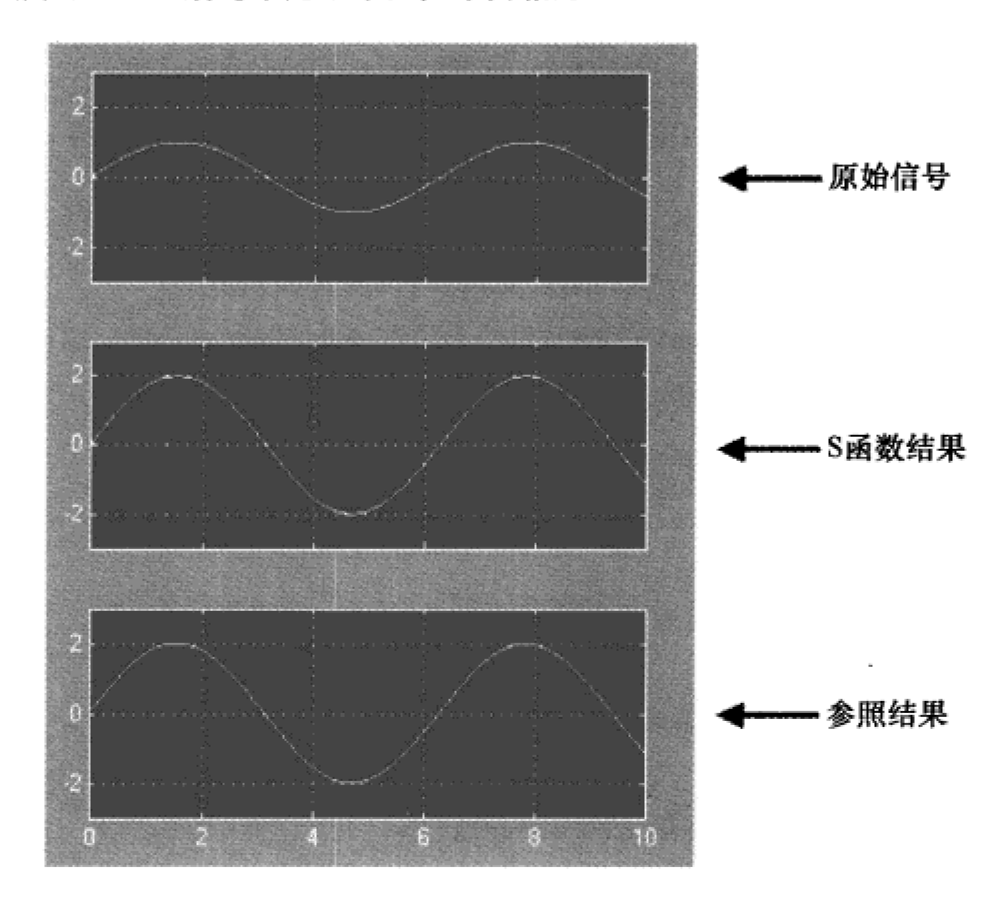


图 4.1.9 输出结果

4.1.3 S-Function Builder

下面仍然以 doubleIt. c 为例，使用 S-Function Builder 整合 doubleIt. c，创建一个 S-Function Builder 模块。

为了对比 S-Function Builder 创建模块的功能，按照图 4.1.10 新建一个包含 S-Function Builder 的 Simulink 模型。Gain 模块完成乘以 2 的功能，验证 S-Funtion Builder 能否实现 Gain 模块相同的功能，Sine Wave 的幅值设为 1。

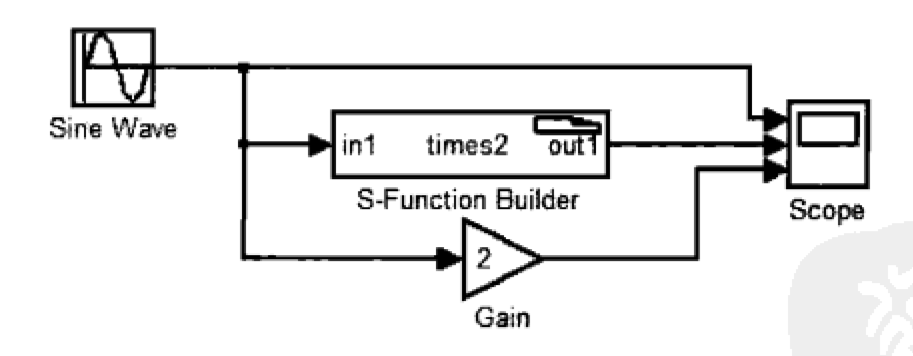


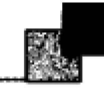
图 4.1.10 S-Function Builder 模型

双击打开 S-Function Builder 模块，作以下设置：

(1) 在 S-function name 字段中输入 times2。

(2) 初始化界面在本例中不需要更改。

(3) 在 Data Properties 中，将输入/输出定义为 in1、out1，如图 4.1.11、图 4.1.12 所示。



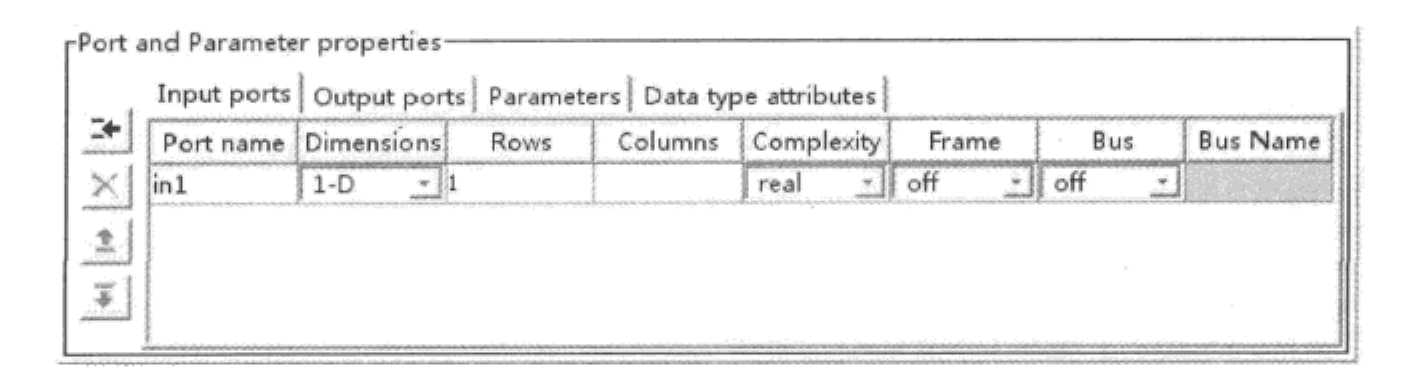


图 4.1.11　Input Ports 界面

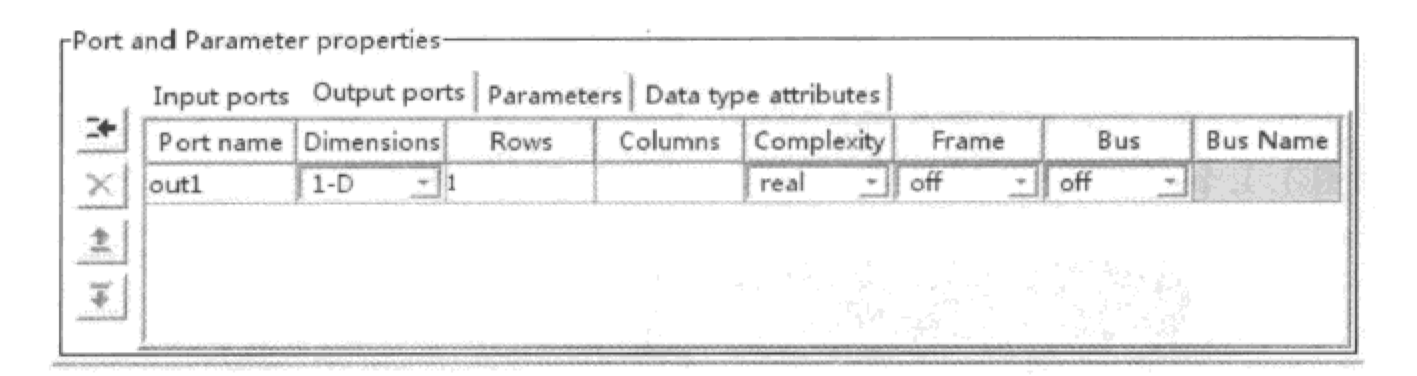


图 4.1.12　Output Ports 界面

(4) 在库文件界面中的 Library/Object/Source files 区域输入 doubleIt. c，在 Includes 区域输入 #include "doubleIt. h"，如图 4.1.13 所示。

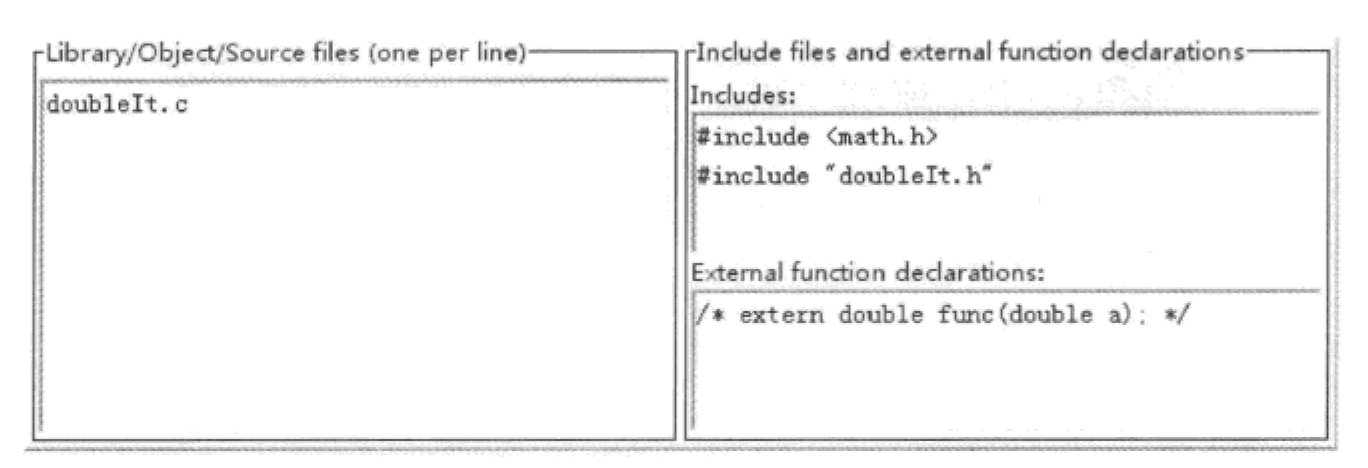


图 4.1.13　库文件界面

(5) 在输出界面中输入 *out1 = doubleIt(*in1)，如图 4.1.14 所示。

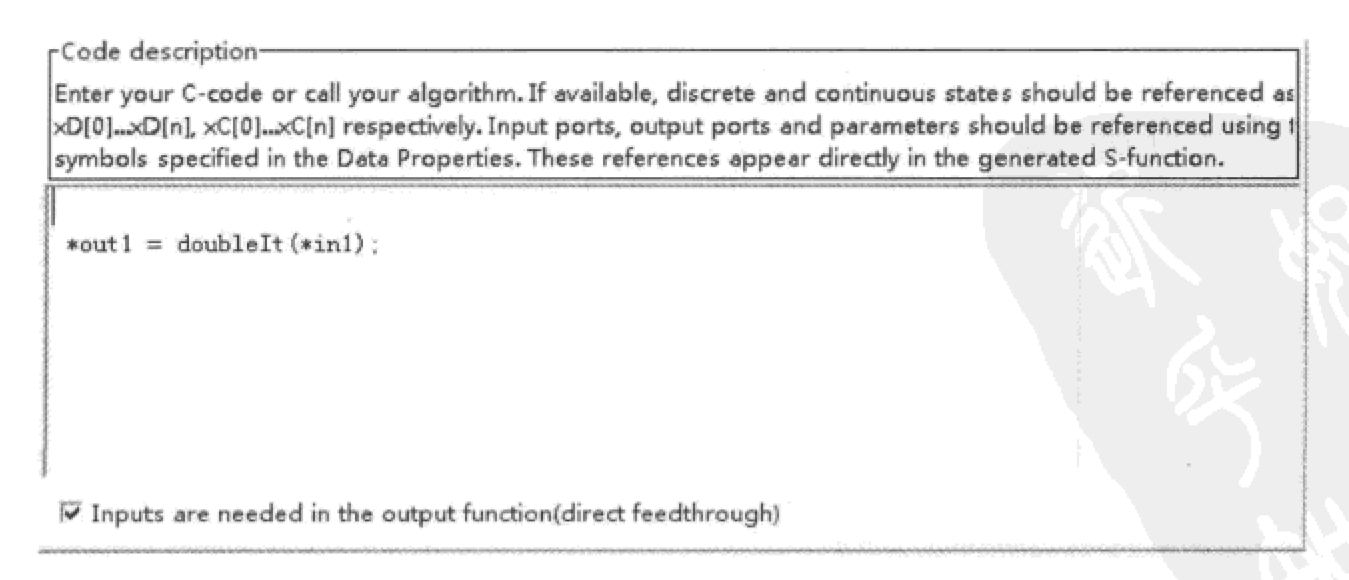


图 4.1.14　输出界面

（6）单击对话框工具栏的 Build 按钮，编译信息窗口即显示完成信息，如图 4.1.15 所示。

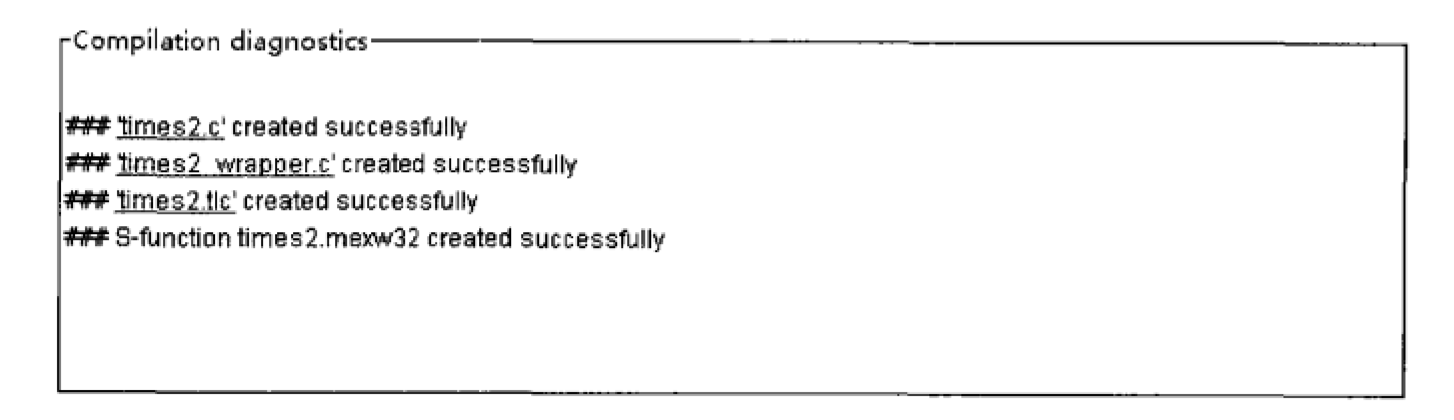


图 4.1.15　编译信息

按下“仿真”按钮，示波器显示 S 函数输出结果与增益模块的一致，说明 S 函数实现了设计的功能，如图 4.1.16 所示。

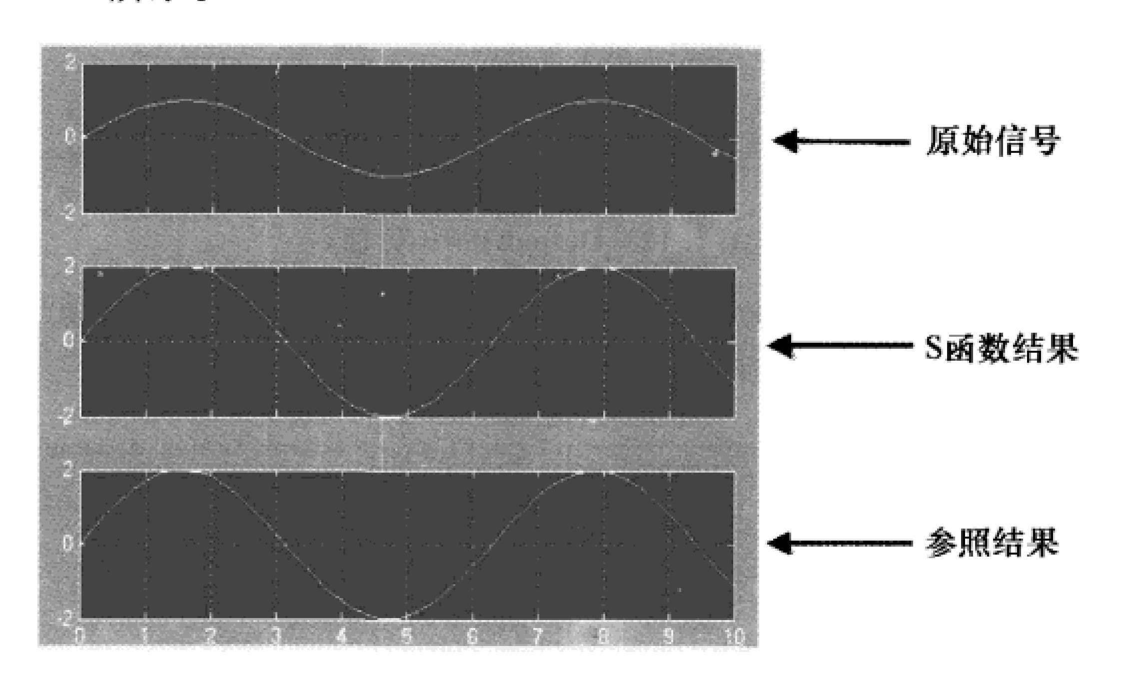


图 4.1.16　仿真结果

4.1.4　三种方法的比较

由 LCT 工具生成的 S 函数与 S-Function Builder 生成的不同之处在于：

（1）由 S-Function Builder 生成的 S 函数通过 wrapper 函数的 builder_wrapsfcn_wrapper.c 调用 doubleIt.c；而由 LCT 生成的 S 函数直接在 mdlOutputs 回调方法里调用 doubleIt.c。

（2）S-Function Builder 允许用户在 Data Properties 面板里定义 S 函数的输入/输出名；而 LCT 则使用默认的 u 和 y 作为输入/输出名，用户无法更改。

（3）S-Function Builder 与 LCT 默认情况下，都使用继承的采样时间，但 S-Function Builder 的偏移时间为 0.0，LCT 则固定为次仿真步长。

表 4.1.1 列出了手写 S 函数、代码继承工具 LCT 以及 S-Function Builder 三种方法之间的异同。

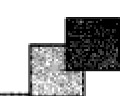


表 4.1.1　三种生成 S 函数方法的异同

类　型	手写 C MEX S 函数	代码继承工具 LCT	S-Function Builder
数据类型	支持所有的 Simulink 数据类型，包括定点数据	支持所有内建的数据类型 如果要使用定点数据，必须将其指定为 Simulink. NumericType，如果用户未指定范围，则不能使用	支持所有的 Simulink 数据类型，包括定点数据
数值类型	支持实数与复数信号	支持内建数据类型的复数信号	支持实数与复数信号
帧信号	支持帧信号与非帧信号	不支持帧信号	支持帧信号与非帧信号
端口维度	支持标量、一维、多维的输入/输出信号	支持标量、一维、多维的输入/输出信号	支持标量、一维、多维的输入/输出信号
S-function API	支持全部的 S-function API	支持以下回调方法： mdlInitializeSizes mdlInitializeSampleTimes mdlStart mdlInitializeConditions mdlOutputs mdlTerminate	支持以下回调方法： mdlInitializeSizes mdlInitializeSampleTimes mdlStart mdlDerivative mdlUpdate mdlOutput mdlTerminate
代码生成支持	支持代码生成，如果在代码生成时需要内嵌 S 函数，则需要手写 TLC 文件	支持针对嵌入式优化的代码生成，同时可自动地生成 TLC 文件。该 TLC 文件支持在代码生成时内嵌 S 函数的表达式折叠	支持代码生成，同时自动地生成 TLC 文件，用于代码生成时内联 S 函数
Simulink Accelerator 模式	提供了在 Accelerator 模式下使用 TLC 或 MEX 文件的选项	提供了在 Accelerator 模式下使用 TLC 或 MEX 文件的选项	如果生成了 TLC 文件，则在 Accelerator 模式下使用该文件，否则使用 MEX 文件
模型引用	使用引用模型时，提供了继承采样时间选项与 Normal 模式支持	使用引用模型时，使用默认的行为	使用引用模型时，使用默认的行为
Simulink. AliasType，Simulink. NumericType Simulink. StructType 类型支持	支持所有的类，其中 Simulink. StructType 仅用于 S 函数参数	支持的类有： Simulink. AliasType Simulink. NumericType	不支持这些类
输入/输出总线信号	不支持输入/输出总线信号	支持输入/输出总线信号，但用户需要事先在 MATLAB 工作空间定义 Simulink. Bus 对象，该对象必须与原始 C 代码的输入输出结构等价 不支持总线参数	不支持输入/输出总线信号

续表 4.1.1

类　型	手写 C MEX S 函数	代码继承工具 LCT	S-Function Builder
可调参数与实时参数	支持可调参数与实时参数	支持可调参数与实时参数	支持实时参数，仅在仿真时支持可调参数
工作向量	支持所有的工作向量类型	使用 SS_DWORK_USED_AS_DWORK 类型时，支持 DWork 工作向量	无法访问工作向量

表 4.1.2 进一步总结了上述三种方法的主要功能限制

表 4.1.2　三种生成 S 函数方法的限制

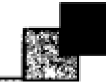
	限　制
手写 C MEX S 函数	(1) 不支持总线输入/输出信号 (2) 不支持模型引用
代码继承工具 LCT	(1) 支持使用 C、C++编写的源代码，不支持 MATLAB 与 Fortran 函数 (2) 支持 C++函数、但不支持 C++对象 (3) 不支持连续及离散状态仿真 (4) 不支持现有 C 代码使用函数指针作为输出 (5) 直馈标志位始终为真 (6) 仅支持连续求解器，但次仿真步长、采样时间及偏移时间是固定的 (7) 仅支持 Simulink 内建数据类型的复数 (8) 不支持工作向量(除了通用 DWork 向量) (9) 不支持帧信号输入/输出 (10) 不支持基于端口的采样时间 (11) 不支持基于多模块的采样时间
S-Function Builder	(1) 目标 S 函数使用 wrapper 函数来实现，但其中包含多余的开销 (2) 不支持工作向量 (3) 不支持基于端口的采样时间 (4) 不支持多采样时间，以及非零的偏移时间 (5) 不支持多输入/输出口的动态信号，但对于单输入/输出口的情况，支持动态信号

4.2 S 函数

4.2.1 S 函数工作机制

创建 S 函数之前，必须要了解它的运行机制，而这又需要事先了解 Simulink 模型的仿真过程。

对于 Simulink 模块，一般包含输入变量、状态变量及输出变量，其中输出是仿真时间、输



入变量、状态变量的函数。如图4.2.1所示。



图4.2.1 输入、状态和输出关系图

其数学表达式概括如下：

$y=f_0(t,x,u)$ 输出

$x'=f_d(t,x,u)$ 微分

$x_{d+}=f_u(t,x_c,x_d u)$ 离散状态更新

其中 $x=[x_c,x_d]$

Simulink模型的工作流程，是通过回调方法实现的，如图4.2.2所示。在仿真的各个阶段，Simulink引擎调用不同的方法，执行各项任务。

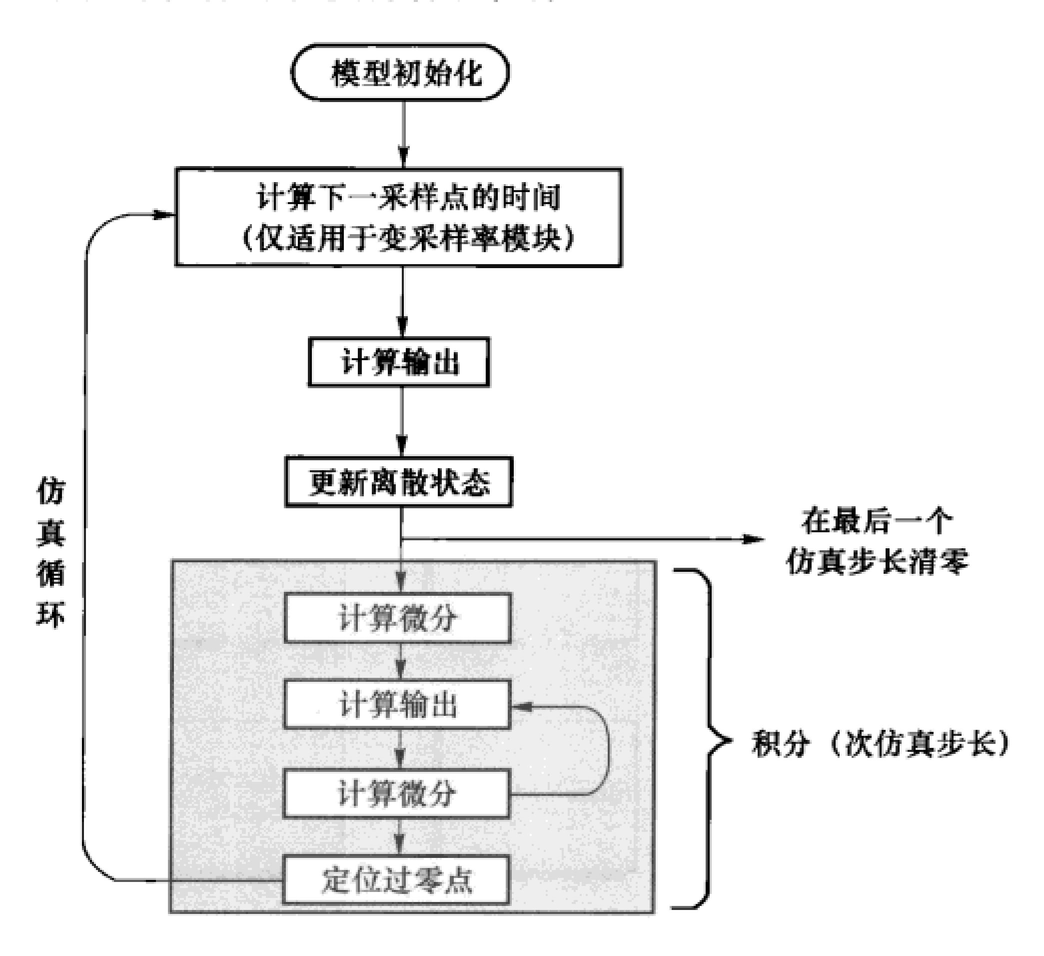


图4.2.2 模型工作流程

1. 初始化阶段

在第一次仿真循环之前，Simulink引擎初始化S函数：

(1) 初始化结构体SimStruct，它包含了S-Function的所有信息。

(2) 设置输入/输出端口数与维度。

(3) 设置模块采样时间个数。

(4) 决定模块的执行次序并分配存储空间。

2. 仿真循环阶段

一次循环可称为一个仿真步长，在每一仿真步长内，系统按照初始化阶段决定的次序依次执行各模块。对于每个模块，系统调用各种函数计算当前采样时间的模块状态、微分值以及模块输出，它又可以分为以下几个步骤：

(1) 计算下一个采样时间点。对于变采样时间的模块,Simulink 引擎需要调用该方法,计算下一个采样时间。

(2) 计算输出。在主仿真步长,计算所有模块的输出值。

(3) 更新离散状态。在主仿真步长,更新所有的离散状态。

(4) 数值积分。该积分循环仅用于包含连续状态和/或非采样过零点的模型。

如果 S 函数存在连续状态,Simulink 引擎在次仿真步长内调用微分和输出两个回调方法。如果 S 函数存在非采样过零点,Simulink 引擎在次仿真步长内调用输出和过零检测函数。不断执行该循环,可以得到状态所需要的精度。

4.2.2 C MEX S 函数模板

Mathworks 公司提供了两种 C MEX S 函数模板:

简化模板:位于...\MATLAB\R2010b\simulink\src\sfuntmpl_basic.c。

详细模板:位于...\MATLAB\R2010b\simulink\src\sfuntmpl_doc.c。

用户还可以在 Simulink 模块浏览器的 Simulink→User-Defined Functions→S-function Examples→C file S-functions 子模块库找到这两个模板,如图 4.2.3 所示。

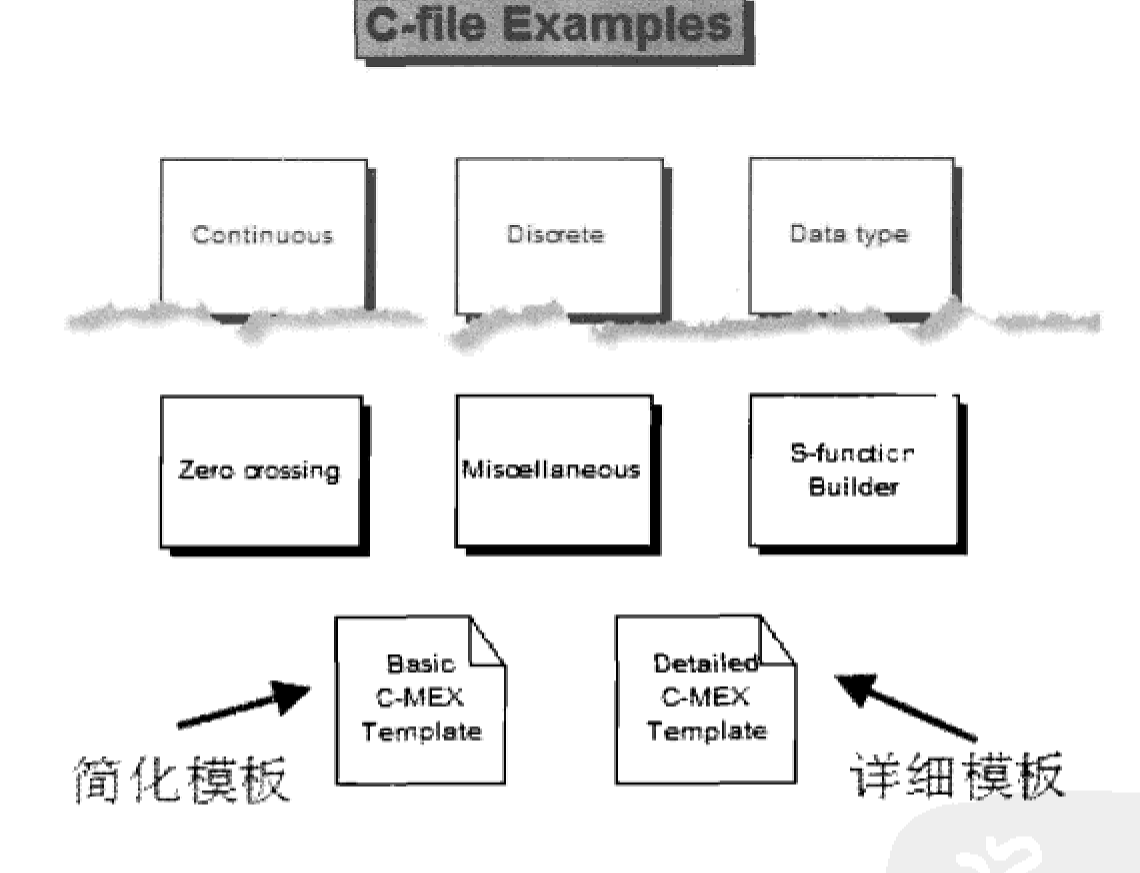
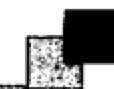


图 4.2.3 C MEX S 函数模板

在 4.1 节中,用户已初步认识了 S 函数,本小节以分块的形式介绍 S 函数的组成,为保持模板的完整性,在代码段内不作任何中文说明,因此将这些代码段组合起来,就是一个完整的简化模板。

1. 文件头部

```
/*
 * sfuntmpl_basic.c: Basic 'C' template for a level 2 S-function.
 * -------------------------------------------------------------------------
```



```
 * | See matlabroot/simulink/src/sfuntmpl_doc.c for a more detailed template |
 * -------------------------------------------------------------------------
 * Copyright 1990-2002 The MathWorks, Inc.
 * $ Revision: 1.27.4.2 $
 */

/*
 * You must specify the S_FUNCTION_NAME as the name of your S-function
 * (i.e. replace sfuntmpl_basic with the name of your S-function).
 */
#define S_FUNCTION_NAME sfuntmpl_basic
#define S_FUNCTION_LEVEL 2
/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"
```

在文件头部,可以定义变量,加入需要的头文件。

模板给出了必要的函数名与函数级别的定义方式,用户应将 sfuntmpl_basic 替换为其他合适的名称,函数级别 2 一般不作更改。

simstruc.h 是必要的头文件,它定义了一个数据结构体 SimStruct 以及大量的宏函数,可供后期调用。

2. 错误处理

```
/* Error handling
 * --------------
 * You should use the following technique to report errors encountered
 * within an S-function:
 *       ssSetErrorStatus(S,"Error encountered due to ...");
 *       return;
 * Note that the 2nd argument to ssSetErrorStatus must be persistent
 * memory.
 * It cannot be a local variable. For example the following will cause
 * unpredictable errors:
 *       mdlOutputs()
       {
 *          char msg[256]; {ILLEGAL: to fix use "static char msg[256];"}
 *          sprintf(msg,"Error due to %s", string);
 *          ssSetErrorStatus(S,msg);
 *          return;
 *       }
 * See matlabroot/simulink/src/sfuntmpl_doc.c for more details.
 */
```

在模板里，错误处理以注释方式表示，这说明它们不是必需的，但并不表示这不重要。

例如，在 S 函数执行之前，加入回调方法 mdlCheckParameters(SimStruct * S)，可以检查输入的参数是否有效，避免了 S 函数使用无效的参数进行计算。

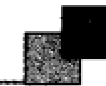
宏函数 ssSetErrorStatus 的第二个参数，必须永久驻留在内存，而不能是某个函数的局部变量，模板给出了该函数的一种错误使用，用户应当避免。

```
void ssSetErrorStatus(SimStruct * S, const char_T * msg)
```

该函数在执行过程中，不会产生异常状态，因此若用户需要进行异常处理，则建议使用 mexErrMsgTxt 作为替代，不过它需要另外建立一个优于所有 S 函数调用的异常处理机制，这将带来额外的开销。第 4.4.1 节，介绍了错误处理的实际应用。

3. 回调方法 mdlInitializeSizes

```
/* S-function methods */
/* Function: mdlInitializeSizes
 * Abstract:
 *    The sizes information is used by Simulink to determine the
 *    S-function block's characteristics (number of inputs,
 *    outputs, states, etc.).
 */
static void mdlInitializeSizes(SimStruct * S)
{
/* See sfuntmpl_doc.c for more details on the macros below */
    ssSetNumSFcnParams(S, 0); /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
/* Return if number of expected != number of actual parameters */
        return;
    }
    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);
    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 1);
    ssSetInputPortRequiredContiguous(S, 0, true);
                                         /* direct input signal access */
    /* Set direct feedthrough flag (1 = yes, 0 = no).
     * A port has direct feedthrough if the input is used in either
     * the mdlOutputs or mdlGetTimeOfNextVarHit functions.
     * See matlabroot/simulink/src/sfuntmpl_directfeed.txt.
     */
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);
```



```
    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
/* Specify the sim state compliance to be same as a built-in block */
    ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);
    ssSetOptions(S, 0);
}
```

mdlInitializeSizes 是必要的回调方法，它用于以下几方面：

(1) 设置 S 函数的参数个数。

(2) 设置 S 函数的状态数量。

(3) 设置输入口的个数、维度以及直馈标志位。

(4) 设置输出口的个数、维度。

(5) 设置采样率的个数。

(6) 设置模块工作参数的大小。

(7) 设置模块仿真选项。

这些工作通常由宏函数完成，为了便于理解，表 4.2.1 列出了代码中的宏函数意义，完整的函数使用方法，请参考文献。

表 4.2.1　mdlInitializeSizes 宏函数意义

宏函数	意　义
ssSetNumSFcnParams(S, 0)	设置 S 函数参数的个数为 0
ssGetNumSFcnParams(S)	获取 S 函数参数的个数
ssGetSFcnParamsCount(S)	获取封装 S 函数时，对话框的参数个数
ssSetNumContStates(S, 0)	设置 S 函数的连续状态数量为 0
ssSetNumDiscStates(S, 0)	设置 S 函数的离散状态数量为 0
ssSetNumInputPorts(S, 1)	设置 S 函数的输入口数量为 1
ssSetInputPortWidth(S, 0, 1)	设置 S 函数第 0 号输入口的维度为 1(端口的次序从 0 开始编排)
ssSetInputPortRequiredContiguous(S, 0, true)	设置 S 函数第 0 号输入口的信号保存在连续的内存空间
ssSetInputPortDirectFeedThrough(S, 0, 1)	设置 S 函数第 0 号输入口的直馈状态为真
ssSetNumOutputPorts(S, 1)	设置 S 函数的输出口个数为 1
ssSetOutputPortWidth(S, 0, 1)	设置 S 函数第 0 号输出口的宽度为 1
ssSetNumSampleTimes(S, 1)	设置 S 函数的采样时间个数为 1
ssSetNumRWork(S, 0)	设置 S 函数的浮点工作向量的大小为 0
ssSetNumIWork(S, 0)	设置 S 函数的整型工作向量的大小为 0
ssSetNumPWork(S, 0)	设置 S 函数的指针工作向量的大小为 0

续表 4.2.1

宏函数	意　义
ssSetNumModes(S, 0)	设置 S 函数的模式向量的大小为 0
VssSetNumNonsampledZCs(S, 0)	设置 S 函数在采样点之间检测过零状态的数量为 0
ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE)	设置 S 函数在保存及恢复模型仿真状态时的行为
ssSetOptions(S, 0)	设置 S 函数的选项为无

4. 回调方法 mdlInitializeSampleTimes

```
/* Function: mdlInitializeSampleTimes
 * Abstract:
 *     This function is used to specify the sample time(s) for your
 *     S-function. You must register the same number of sample times as
 *     specified in ssSetNumSampleTimes.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S, 0, 0.0);
}
```

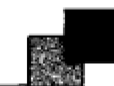
mdlInitializeSampleTimes 也是必要的回调方法，用于设置采样时间值和偏移量，采样时间个数必须与宏函数 ssSetNumSampleTimes 设置的一致。

表 4.2.2　mdlInitializeSampleTimes 宏函数意义

宏函数	意　义
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME)	设置第 1 个采样时间的周期为连续采样
ssSetOffsetTime(S, 0, 0.0)	设置第 1 个采样时间的偏移量

5. 回调方法 mdlInitializeConditions

```
#define MDL_INITIALIZE_CONDITIONS
/* Change to #undef to remove function */
#if defined(MDL_INITIALIZE_CONDITIONS)
/* Function: mdlInitializeConditions
     * Abstract:
     *     In this function, you should initialize the continuous and
     *     discrete states for your S-function block.
     *     The initial states are placed in the state vector,
     *     ssGetContStates(S) or ssGetRealDiscStates(S).
     *     You can also perform any other initialization activities
     *     that your S-function may require.
     *     Note, this routine will be called at the
```



```
 *     start of simulation and if it is present in an enabled subsystem
 *     configured to reset states, it will be call when the enabled
 *     subsystem restarts execution to reset the states.
 */
static void mdlInitializeConditions(SimStruct *S)
{
}
#endif /* MDL_INITIALIZE_CONDITIONS */
```

mdlInitializeConditions 是一个可选的回调方法，如果用户定义了连续或离散状态，则必须在此初始化这些状态。当然这里也可以定义其他的初始化操作。

6. 回调方法 mdlStart

```
#define MDL_START /* Change to #undef to remove function */
#if defined(MDL_START)
  /* Function: mdlStart
   * Abstract:
   *     This function is called once at start of model execution. If
   *     you have states that should be initialized once,
   *     this is the place to do it.
   */
  static void mdlStart(SimStruct *S)
  {
  }
#endif /* MDL_START */
```

mdlStart 也是一个可选的回调方法。用于在仿真开始时，为 S 函数分配内存，设置用户数据，初始化状态。该方法只执行一次，因此如果 S 函数位于一个使能子系统内，而用户又希望在每次子系统有效时，重新初始化状态，则需要把该任务定义在 mdlInitialize Conditions。

7. 回调方法 mdlOutputs

```
/* Function: mdlOutputs
 * Abstract:
 *     In this function, you compute the outputs of your S-function
 *     block.
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    const real_T *u = (const real_T*) ssGetInputPortSignal(S,0);
    real_T       *y = ssGetOutputPortSignal(S,0);
    y[0] = u[0];
}
```

mdlOutputs 是一个必要的回调方法，在每个仿真步长，计算当前步长的 S 函数输出，并将

结果保存在 S 函数输出信号数组。第 4.4.1 节，介绍了输出回调方法的实际应用。

表 4.2.3 一组常用的信号访问宏函数

宏函数	意 义
ssGetInputPortRealSignal	获取输入连续实数信号所在的内存地址
ssGetInputPortRealSignalPtrs	获取 double 类型输入信号的指针
ssGetInputPortSignal	获取输入连续信号数据所在的内存地址
ssGetInputPortSignalPtrs	获取除了 double 类型输入信号的指针
ssGetOutputPortRealSignal	获取 double 类型输出信号的指针
ssGetOutputPortSignal	获取输出信号向量的指针

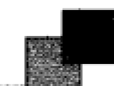
8. 回调方法 mdlUpdate

```
#define MDL_UPDATE  /* Change to #undef to remove function */
#if defined(MDL_UPDATE)
  /* Function: mdlUpdate
   * Abstract:
   *     This function is called once for every major integration time
   *     step. Discrete states are typically updated here,
   *     but this functionis useful for performing any tasks
   *     that should only take place once per integration step.
   */
  static void mdlUpdate(SimStruct *S, int_T tid)
  {
  }
#endif /* MDL_UPDATE */
```

mdlUpdate 是一个可选的回调方法，在每个主仿真步长，计算 S 函数的当前状态，将结果保存在状态向量，同时也可执行其他需要在主仿真步长进行的操作。

9. 回调方法 mdlDerivatives

```
#define MDL_DERIVATIVES /* Change to #undef to remove function */
#if defined(MDL_DERIVATIVES)
  /* Function: mdlDerivatives
   * Abstract:
   *     In this function, you compute the S-function block's
   *     derivatives. The derivatives are placed in the derivative
   *     vector, ssGetdX(S).
   */
  static void mdlDerivatives(SimStruct *S)
  {
  }
#endif /* MDL_DERIVATIVES */
```



mdlDerivatives 是一个可选的回调方法，在每个仿真步长，计算 S 函数中连续状态的微分，并将微分值保存在 S 函数状态微分数组。

10. 回调方法 mdlTerminate

```
/* Function: mdlTerminate
 * Abstract:
 *     In this function, you should perform any actions that are
 *     necessary at the termination of a simulation. For example, if
 *     memory was allocated in mdlStart, this is the place to free it.
 */
static void mdlTerminate(SimStruct *S)
{
}
```

mdlTerminate 是最后一个必要的回调方法，该方法执行诸如释放在 mdlStart 方法中分配的内存等，必须在仿真结束时或 S 函数模块被删除时才能执行的动作。

对于 C MEX S-functions，当用户执行了模型更新（Update Diagram）或系统遇到以下任何一种情况时，调用该 mdlTerminate 方法。

（1）系统调用了 mdlStart 方法，但 S 函数本身未定义任何的 mdlInitializeConditions 方法。

（2）系统调用了 mdlInitializeConditions 方法。

11. 文件尾部

```
/* Required S-function trailer */
#ifdef MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file?
#include "simulink.c"     /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"      /* Code generation registration function */
#endif
```

这些尾部信息是必需的，它们作为 MEX 文件与 Real-Time Workshop 的接口，一般情况下，用户不需作任何修改。

4.2.3 其他回调方法

以下简要介绍其余 27 个可选回调方法的语法及意义：

1. 函数名：mdlCheckParameters

语法：

```
#define MDL_CHECK_PARAMETERS
void mdlCheckParameters(SimStruct *S)
```

意义：检查 S 函数参数的有效性。

2. 函数名：mdlDisable

语法：



```
#define MDL_DISABLE
void mdlDisable(SimStruct *S)
```

意义：如果一个有效的子系统在当前仿真步长，即将由使能状态转换为禁止状态，则 Simulink 引擎调用该方法。

3. 函数名：mdlEnable

语法：

```
#define MDL_ENABLE
void mdlEnable(SimStruct *S)
```

意义：如果一个有效的子系统在当前仿真步长，即将由禁止状态转换为使能状态，则 Simulink 引擎调用该方法。

4. 函数名：mdlGetSimState

语法：

```
#define MDL_SIM_STATE
mxArray* mdlGetSimState(SimStruct* S)
```

意义：该方法返回当前 S 函数的仿真状态，保存为 MATLAB 数据结构体。

5. 函数名：mdlGetTimeOfNextVarHit

语法：

```
#define MDL_GET_TIME_OF_NEXT_VAR_HIT
void mdlGetTimeOfNextVarHit(SimStruct *S)
```

意义：指定下一个仿真时间。

6. 函数名：mdlProcessParameters

语法：

```
#define MDL_PROCESS_PARAMETERS
void mdlProcessParameters(SimStruct *S)
```

意义：处理 S 函数的参数。

7. 函数名：mdlProjection

语法：

```
#define MDL_PROJECTION
void mdlProjection(SimStruct *S)
```

意义：调整系统状态的求解器方案，以更好地满足时不变求解关系。

8. 函数名：mdlRTW

语法：



```
#define MDL_RTW
void mdlRTW(SimStruct *S)
```

意义：在 Real-Time Workshop 工具生成 model.rtw 文件时，调用该方法。

9. 函数名：mdlSetDefaultPortComplexSignals

语法：

```
#define MDL_SET_DEFAULT_PORT_COMPLEX_SIGNALS
void mdlSetDefaultPortComplexSignals(SimStruct *S)
```

意义：如果系统无法正确获得模块所有端口的数值类型（实数、复数、继承），则将其设置为该方法定义的默认类型。如果模块对应的 S 函数未定义该方法，而至少有一个端口的数值类型是复数，则设置其他未知端口为 COMPLEX_YES，否则设置为 COMPLEX_NO。

10. 函数名：mdlSetDefaultPortDataTypes

语法：

```
#define MDL_SET_DEFAULT_PORT_DATA_TYPES
void mdlSetDefaultPortDataTypes(SimStruct *S)
```

意义：如果系统无法正确获得模块所有端口的数据类型，则将其设置为该方法定义的默认类型。如果模块对应的 S 函数亦无定义该方法，则将所有端口设置为 double 类型。若某些端口已有定义，则将其他未定义的端口设置为已定义数据类型中最大的一种。

11. 函数名：mdlSetDefaultPortDimensionInfo

语法：

```
#define MDL_SET_DEFAULT_PORT_DIMENSION_INFO
void mdlSetDefaultPortDimensionInfo(SimStruct *S)
```

意义：如果模型无法提供足够的信息，以建立信号维度，则系统根据该方法定义的默认信息，完成信号维度传递。

12. 函数名：mdlSetInputPortComplexSignal

语法：

```
#define MDL_SET_INPUT_PORT_COMPLEX_SIGNAL
void mdlSetInputPortComplexSignal(SimStruct *S, int_T port, CSignal_T csig)
```

意义：设置输入端口的数值类型（实数、复数、继承）。

13. 函数名：mdlSetInputPortDataType

语法：

```
#define MDL_SET_INPUT_PORT_DATA_TYPE
void mdlSetInputPortDataType(SimStruct *S, int_T port, DTypeId id)
```

意义:设置输入端口的数据类型。

14. 函数名:mdlSetInputPortDimensionInfo

语法:

```
#define MDL_SET_INPUT_PORT_DIMENSION_INFO
void mdlSetInputPortDimensionInfo(SimStruct *S, int_T port, const DimsInfo_T *dimsInfo)
```

意义:设置输入端口的信号维度。

15. 函数名:mdlSetInputPortDimensionsModeFcn

语法:

```
void mdlSetInputPortDimensionsModeFcn(SimStruct *S, int_T portIdx, DimensionsMode_T dimsMode)
```

意义:设置输入端口的维度模式,其中参数 portIdx 表示端口索引号。

16. 函数名:mdlSetInputPortFrameData

语法:

```
#define MDL_SET_INPUT_PORT_FRAME_DATA
void mdlSetInputPortFrameData(SimStruct *S, int_T port, Frame_T frameData)
```

意义:指定输入端口是否能接受帧数据。

17. 函数名:mdlSetInputPortSampleTime

语法:

```
#define MDL_SET_INPUT_PORT_SAMPLE_TIME
void mdlSetInputPortSampleTime(SimStruct *S, int_T port, real_T sampleTime, real_T offsetTime)
```

意义:设置输入端口的采样时间为继承。

18. 函数名:mdlSetInputPortWidth

语法:

```
#define MDL_SET_INPUT_PORT_WIDTH
void mdlSetInputPortWidth(SimStruct *S, int_T port, int_T width)
```

意义:设置输入端口的宽度。

19. 函数名:mdlSetOutputPortComplexSignal

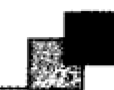
语法:

```
#define MDL_SET_OUTPUT_PORT_COMPLEX_SIGNAL
void mdlSetOutputPortComplexSignal(SimStruct *S, int_T port, CSignal_T csig)
```

意义:设置输出端口的数值类型(实数、复数、继承)。

20. 函数名:mdlSetOutputPortDataType

语法:



```
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
void mdlSetOutputPortDataType(SimStruct *S, int_T port, DTypeId id)
```

意义：设置输出端口的数据类型。

21. 函数名：mdlSetOutputPortDimensionInfo

语法：

```
#define MDL_SET_OUTPUT_PORT_DIMENSION_INFO
void mdlSetOutputPortDimensionInfo(SimStruct *S, int_T port, const DimsInfo_T *dimsInfo)
```

意义：设置输出端口的信号维度。

22. 函数名：mdlSetOutputPortSampleTime

语法：

```
#define MDL_SET_OUTPUT_PORT_SAMPLE_TIME
void mdlSetOutputPortSampleTime(SimStruct *S, int_T port, real_T sampleTime, real_T offsetTime)
```

意义：设置输出端口的采样时间为继承。

23. 函数名：mdlSetOutputPortWidth

语法：

```
#define MDL_SET_OUTPUT_PORT_WIDTH
void mdlSetOutputPortWidth(SimStruct *S, int_T port, int_T width)
```

意义：设置输出端口的宽度。

24. 函数名：mdlSetSimState

语法：

```
#define MDL_SIM_STATE
void mdlSetSimState(SimStruct* S, const mxArray* in)
```

意义：在仿真开始时，设置初始仿真状态为 SimState。

25. 函数名：mdlSetWorkWidths

语法：

```
#define MDL_SET_WORK_WIDTHS
void mdlSetWorkWidths(SimStruct *S)
```

意义：指定工作向量的大小，并建立运行时参数。

26. 函数名：mdlSimStatusChange

语法：

```
#define MDL_SIM_STATUS_CHANGE
void mdlSimStatusChange(SimStruct *S, ssSimStatusChangeType simStatus)
```

意义：在仿真暂停或恢复时，调用该回调方法。

27. 函数名:mdlZeroCrossings

语法:

```
#define MDL_ZERO_CROSSINGS
void mdlZeroCrossings(SimStruct *S)
```

意义:如果用户需要进行过零检测,则需要定义该方法与工作向量。

4.2.4 宏函数

以下选取几个常用的宏函数,详细说明其使用方法。

1. 输入/输出口

多数情况下,输入/输出口的宏函数是成对出现的,因此这里仅介绍输入口。

(1) 函数名 ssSetInputPortWidth

语法

```
int_T ssSetInputPortWidth(SimStruct *S, int_T port, int_T width)
```

作用:设置输入/输出口的宽度;

参数port:端口序号(从 0 开始)。

width:宽度值,必须是非 0 的正整数,或者是"DYNAMICALLY_SIZED",如果设为后者,则需要另行定义 mdlSetInputPortDimensionInfo 与 mdlSetDefaultPortDimensionInfo 两个回调方法。

应用:设置第二个输入口的宽度为 5,则

```
ssSetInputPortWidth(S, 1, 5)
```

(2) 函数名 ssSetInputPortMatrixDimensions

语法

```
int_T ssSetInputPortMatrixDimensions(SimStruct *S, int_T port, int_T m, int_T n)
```

作用:设置输入口能够接受矩阵信号。

参数port:端口序号(从 0 开始)。

m, n:矩阵的两个维度,如果其中某一维度为 DYNAMICALLY_SIZED,则另一个也必须为 DYNAMICALLY_SIZED 或 1。如果设为 DYNAMICALLY_SIZED,则需要另行定义 mdlSetInputPortDimensionInfo 与 mdlSetDefaultPortDimensionInfo 两个回调方法。

应用:设置第一个输入口可接受 128×128 的矩阵信号。

```
ssSetInputPortMatrixDimensions(S, 0, 128, 128)
```

2. 采样时间

(1) 函数名 ssSetNumSampleTimes

语法

```
int_T ssSetNumSampleTimes(SimStruct *S, int_T nSampleTimes)
```



作用:设置 S 函数采样时间的个数。

参数 nSampleTimes:采样时间个数,若设置为大于 0 的正整数,表示是基于模块的采样;若设置为 PORT_BASED_SAMPLE_TIMES,则表示是基于端口的采样。

应用:设置采样时间个数为 2。

```
ssSetNumSampleTimes(S, 2)
```

(2) 函数名 ssSetSampleTime

语法

```
time_T ssSetSampleTime(SimStruct * S, int_T st_index, time_T period)
```

作用:设置指定的采样时间。

参数st_index:采样时间的序号(从 0 开始);

period:采样时间的周期值。

应用:设置第 2 个采样时间的周期为 0.1。

```
ssSetSampleTime(S, 1, 0.1)
```

(3) 函数名 ssSetOffsetTime

语法

```
time_T ssSetOffsetTime(SimStruct * S, int_T st_index, time_T offset)
```

作用:设置指定采样时间的偏移量,在此之前,必须先使用 ssSetSampleTime 设置具体的采样时间。若不设置,则系统将采样时间设为连续,并忽略此时定义的偏移量。

参数st_index:采样时间的序号(从 0 开始)。

offset:偏移量的数值。

应用:设置第 2 个采样时间的偏移量为 0.02。

```
ssSetOffsetTime(S, 1, 0.02)
```

4.2.5　数据访问

一个 S 函数,通常含有输入信号、输出信号、参数、内部状态,以及数值类型、数据类型、指针工作向量等其他属性。

一般来说,模块的输入/输出读写都是通过模块 I/O 向量进行的,可以是通过根输入模块由外部输入或通过根输出模块向外输出,并且接地也可以作为输入信号。

S 函数访问输入信号的方式有两种:指针和相邻输入信号。

1. 指　针

为了便于说明,将 4.1.1 节手写 S 函数的输出部分列于下方:

```
static void mdlOutputs(SimStruct * S, int tid){
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T * y = ssGetOutputPortRealSignal(S,0);
    * y = doubleIt( * uPtrs[0]);
```

```
}
```

代码第二行用 InputRealPtrsType 定义了一个特殊的指针向量 uPtrs，宏函数 ssGetInputPortRealSignalPtrs 的返回值是指向输入信号的指针数组，该数组的每个元素事实上是输入信号所在的地址。因此可以使用表达式 * uPtrs[element]，访问这些信号。

图 4.2.4 所示描述了上述关系。

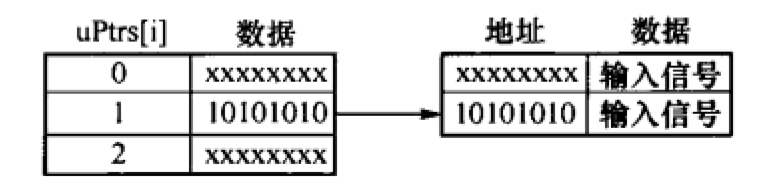


图 4.2.4　输入信号访问方式

当 S 函数有多个输入口时，宏函数 ssGetInputPortRealSignalPtrs 针对每个输入信号返回一个指针数组，这些数组所指向的地址可以是不连续的，图 4.2.5 所示为基于模型的设计及其嵌入式实现。

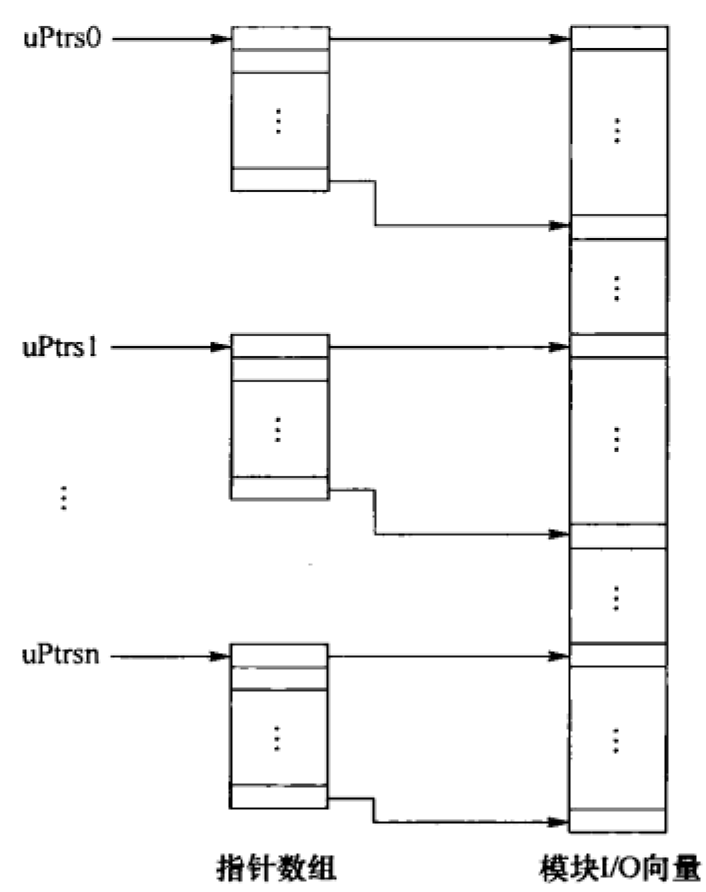


图 4.2.5　访问多输入端口的信号

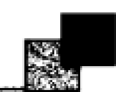
```
InputRealPtrsType uPtrs0  =  ssGetInputPortRealSignalPtrs(S,0);
InputRealPtrsType uPtrs1  =  ssGetInputPortRealSignalPtrs(S,1);
...
InputRealPtrsType uPtrsn  =  ssGetInputPortRealSignalPtrs(S,n - 1);
```

代码第三行定义了输出变量指针，宏函数 ssGetOutputPortSignal 返回输出口的向量，此后再调用外部函数，即可实现源函数的功能。

```
real_T * y  =  ssGetOutputPortRealSignal(S,0);
* y  =  doubleIt( * uPtrs[0]);
```

2. 相邻输入信号

在回调方法 mdlInitializeSizes 中，使用宏函数 ssSetInputPortRequiredContiguous 指定输



入信号的元素占据存储器中的相邻单元。如果输入已经是相邻信号，则用回调方法 ssGetInputPortSignal 访问该输入信号。

以下是两种访问输入信号的常见错误，这些代码虽然可以编译，但 MEX 文件会与 Simulink 冲突，因为它可能会访问无效的内存空间。

```
real_T *u = *uPtrs; /* 不正确 */
*y++ = *u++;   /* 不正确 */
```

4.2.6 目标语言编译器

1. TLC 概述

Target Language Compiler(TLC)是 Real-Time Workshop(RTW)代码生成工具的一个组成部分，它拥有类似于 HTML 的标记语法，类似于 Perl 等其他脚本语言的灵活性，类似于 MATLAB 强大的数据处理能力(TLC 可以调用 MATLAB 函数)。

TLC 包含两组 TLC 文件：

(1) 模块 TLC 文件：它与 Simulink 模块库中的基本模块一一对应，用于生成这些模块的实时代码(不包括 MATLAB function 模块与调用了 MATLAB 文件的 S 函数模块)，用户也可以编写自定义设备驱动模块的 TLC 文件，定制其嵌入式 C 代码的生成过程。

(2) 系统 TLC 文件：用于指定头信息与参数信息，例如 ert.tlc、grt.tlc 等。

合理地编写或修改 TLC 文件，可以定制生成的代码，例如为生成的代码指定硬件平台、合并现有的算法、修改系统目标文件指定的选项、内联 S 函数、生成附加的或其他形式的文件等。这种开放式的环境为用户提供了极大的便利，但用户不应该修改 matlabroot / rtw / c / tlc 目录下的 TLC 文件，这会导致无法预测的结果。

Target Language Compiler 中的 target 不仅表示它将生成高级语言代码，还表示该代码是针对某一实时系统的。这也就说明由 TLC 生成的代码已充分考虑了目标处理器的功能限制，并尽可能地发挥其处理能力。

图 4.2.6 是 TLC 与 RTW 协同工作，生成可执行文件的过程。

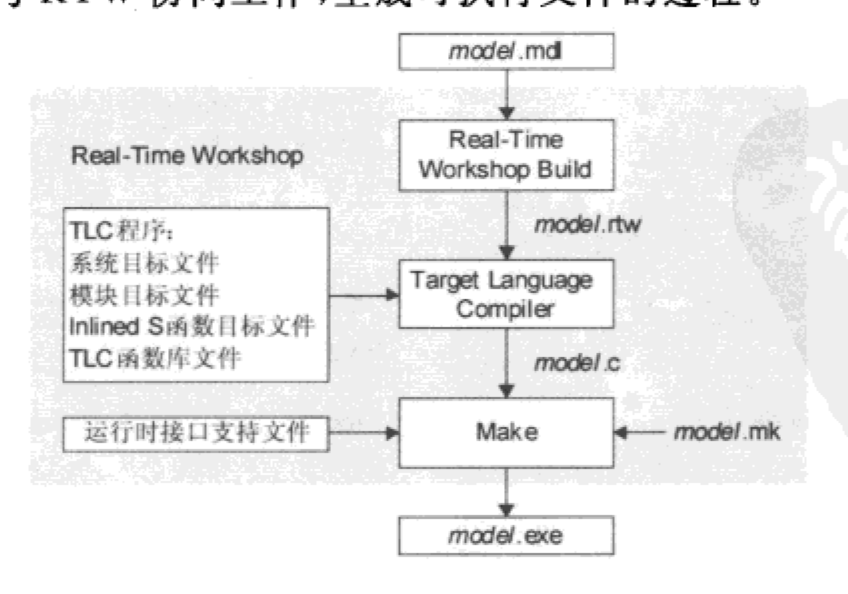


图 4.2.6 可执行文件生成过程

(1) RTW 读取模型文件 model. mdl,生成模型描述文件 model. rtw,该文件以 ASCII 码存储。

(2) TLC 读取 model. rtw,并根据事先选择的系统 TLC 与模块 TLC 文件,生成指定目标的代码,如 ANSI C /C++代码。

(3) RTW 代码生成器根据给定的模型,将 makefile 模板生成具体的 makefile 文件(model. mk),该文件指定了用于生成可执行文件的 C/C++编译器及编译选项。

(4) 连接 model. mk 与运行时接口支持文件,将代码编译生成可执行文件。

2. inlined S 函数的 TLC 文件

S 函数可分为 noninlined S 函数与 inlined S 函数,前者只用于仿真,不需要 TLC 文件,后者用于生成实时代码,需要编写 TLC 文件,同时还删除了对 S 函数自身的调用以及 simstruc. h中一些不必要的部分,加速了系统的仿真以及降低内存的使用(simstruc. h 大约占用 1KB 的内存)。inlined S 函数,还可分为 Fully inlined S 函数与 Wrapper inlined S 函数。这两种 S 函数对应 TLC 文件的写法是不同的。

以下是这两种 S 函数对应的 TLC 文件的示例代码:

```
%% Fully inlined S-functions
% function Outputs(block, system) Output
    %<y> = 2.0 * %<u>;
%endfunction
```

```
%% Wrapper inlined S-functions
% function Outputs(block, system) Output
    %<y> = doubleIt( %<u> );
%endfunction
```

由此可以看出,wrapper TLC 的优势在于:

(1) S 函数与生成代码可共享现有 C 代码,不用重写算法。

(2) 现有的 C 函数是经过优化的程序。

(3) 采用函数调用,使生成的代码更加有效。

(4) 可以将已有的 C 代码无缝地合并到自动生成的 C 代码中。

图 4.2.7 显示了两种 TLC 文件生成代码的区别。

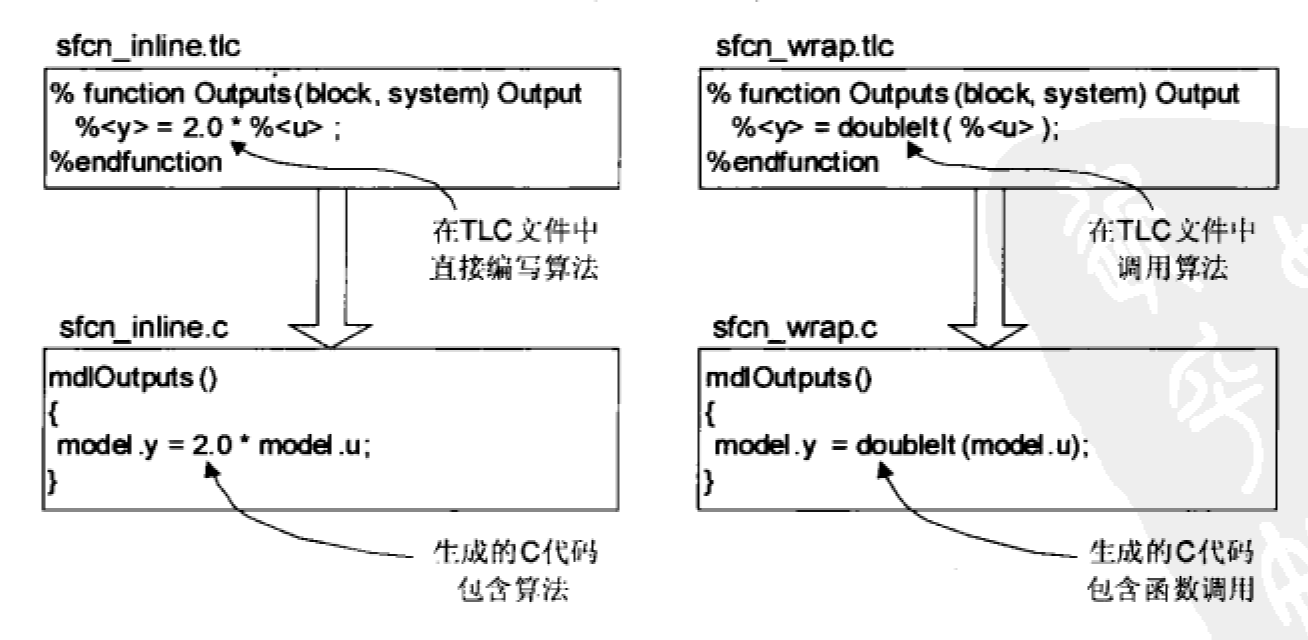
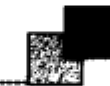


图 4.2.7 两种代码的区别



3. TLC 文件结构

这里仅以 4.1.1 节的例子说明 TLC 文件的结构，代码如下

```
%implements legacy_wrapsfcn "C"
%% Function: BlockTypeSetup
%function BlockTypeSetup(block, system) void
  %% The Target Language must be C
  %if ::GenCPP==1          %% 定义全局变量
    %<LibReportFatalError("This S-Function generated by the Legacy Code Tool must be only used
   with the C Target Language")>
  %endif
  %<LibAddToCommonIncludes("doubleIt.h")>
  %<LibAddToModelSources("doubleIt")>
%endfunction
```

```
%% Function: BlockInstanceSetup
%function BlockInstanceSetup(block, system) void
  %<LibBlockSetIsExpressionCompliant(block)>
%endfunction
%% Function: Outputs
%function Outputs(block, system) Output
    %if !LibBlockOutputSignalIsExpr(0)
    %assign u1_val = LibBlockInputSignal(0, "", "", 0)
    %assign y1_val = LibBlockOutputSignal(0, "", "", 0)
    %<y1_val> = doubleIt( %<u1_val>);
    %endif
%endfunction
```

```
%% Function: BlockOutputSignal
%function BlockOutputSignal(block,system,portIdx,ucv,lcv,idx,retType) void
  %assign u1_val = LibBlockInputSignal(0, "", "", 0)
  %assign y1_val = LibBlockOutputSignal(0, "", "", 0)
  %switch retType
    %case "Signal"
      %if portIdx == 0
        %return "doubleIt( %<u1_val>)"
      %else
        %assign errTxt = "Block output port index not supported: %<portIdx>"
        %<LibBlockReportError(block,errTxt)>
      %endif
    %default
      %assign errTxt = "Unsupported return type: %<retType>"
      %<LibBlockReportError(block,errTxt)>
```

```
  % endswitch
% endfunction
```

TLC 文件的指令以非空格符开始，一般是以字符“%”开始，以“%%”开始的为单行注释。所有的 TLC 文件表达式都要写在<>中：

(1) %implements：是所有模块目标文件必需的，是模块目标文件的第一条可执行指令。

(2) %function：声明一个函数。例如，%function Outputs(block, system) Output，表示定义了一个函数名为 Outputs 的函数。

(3) % openfile、% closefile：建立一个文字缓存区，指令之间的文字，存放在变量 buffer 中。

(4) block：告诉 TLC 是在模块对象上执行的操作，S-Function 的 TLC 代码使用该变量。

(5) %assign 创建或修改变量，U 指所有的模块输入，Y 指所有的模块输出。

(6) 函数 LibBlockInputSignal(portIdx, ucv, lcv, sigIdx)：返回对应参考模块的输入信号。

(7) 函数 LibBlockOutputSignal (portIdx, ucv, lcv, sigIdx)：返回对应参考模块的输出信号。

(8) portIdx：端口索引，从 0 开始编排，0 即表示第 1 个端口。

(9) ucv：用户控制变量，它优先于 lcv 与 sigldx 这两个参数，在使用 inlined S-function 时，该参数应该留空。

(10) lcv：循环控制变量，一般设置 lcv = RollThreshold，RollThreshold 是全局阈值，默认值为 5。sigIdx：信号索引，有些时候也可以认为是信号元素的索引。当需要直接访问输入/输出信号的某一特定元素时，ucv 与 lcv 应留空。sigldx 的值从 0 开始编排。

(11) 若某条指令太长需要另起一行时，可以用 C 语言符号“\”或 MATLAB 中用到的“…”续行，将指令分成多行。例如：

```
%roll sigIdx = RollRegions, lcv = RollThreshold, block, \
"Roller", rollVars
```

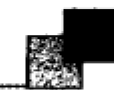
或者

```
%roll sigIdx = RollRegions, lcv = RollThreshold, block, ...
"Roller", rollVars
```

4.3 S-Function Builder

4.3.1 S-Function Builder 简介

MathWorks 公司为了简化 S-Function 模块的编写，提供了 S-Function Builder 模块，用户只要熟悉其面板的意义，按要求一步步做下去就能快速生成用户算法或设备驱动模块。由于 S-Function Builder API 函数只是 S-Function API 的一个子集，所以应用受到一定限制。不过，对于不太熟悉 S-Function 的用户，先从 S-Function Builder 入手，不失为一个好的选择。



S-Function Builder 模块位于 User-Defined Functions→S-Function Builder，如图 4.3.1 所示。

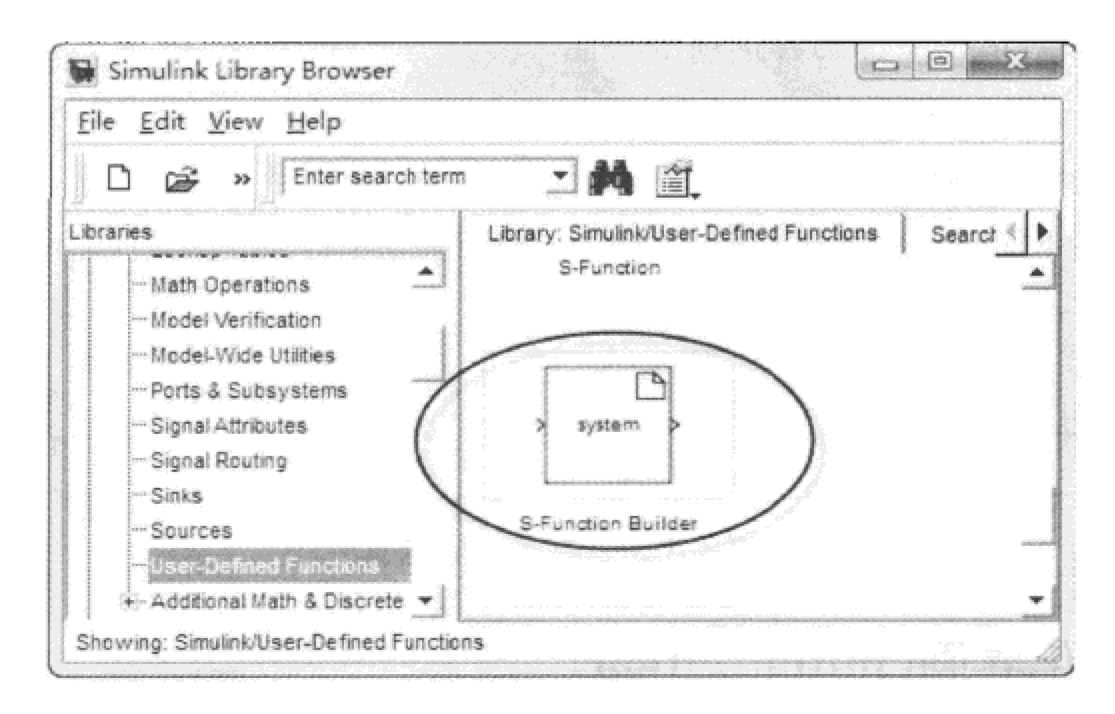


图 4.3.1　S-Function Builder 模块

用户可使用 S-Function Builder 的图形界面设置模块的参数、I/O 接口、状态等，并自动生成一个 S-Function，完成与手写 S-Function 类似的功能。

双击 S-Function Builder 模块，打开 S-Function Builder 对话框，如图 4.3.2 所示。



图 4.3.2　S-Function Builder 对话框

在对话框上部的 S-function name 区域输入用户所指定的 S-Function 名称。S-function parameters 区域列出了该 S-Function 所包含的参数，Name/Data type 的设置将在数据属性界面中介绍，Value 栏用来指定对应参数的参数值，用户可以根据需要在此输入 MATLAB 表达式。设

置完毕后，单击 Builder 按钮即可生成生成 C 源代码与可执行的 MEX 文件，如图4.3.3所示。

以下分节说明对话框下部的各选项卡功能。

Parameters
S-function name: add　Build
S-function parameters
Name | Data type | Value

图 4.3.3　parameters 区域

4.3.2　初始化界面(initialization)

初始化界面 S 函数的设置，如图 4.3.4 所示。

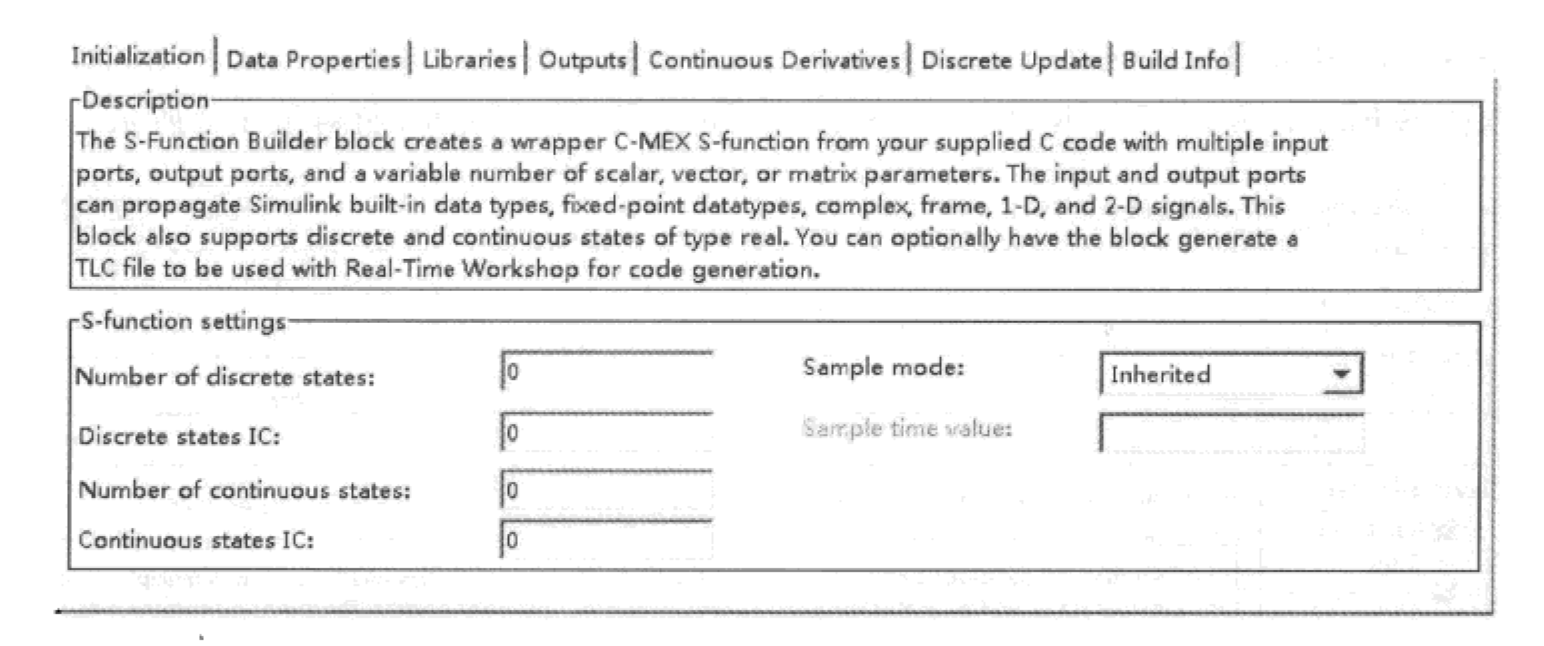


图 4.3.4　初始化界面

(1) Number of discrete states/Number of continuous states：S-Function 中离散或连续状态的数量。

(2) Discrete states IC/Continuous states IC：S-Function 中离散或连续状态的初始状态，用户可以用半角逗号分隔各个初始值，如 0、1、2，或以向量形式表示，如[0 1 2]。初始值的个数必须与先前指定离散或连续状态的数量一致。

(3) Sample mode：

● Inherited：S-Function 模块从与其输入端口相连的前级模块继承采样时间。
● Continuous：以模型的仿真步长作为采样时间。
● Discrete：以下方 Sample time value 文本框指定的数值作为采样时间。

(4) Sample time value：在采样模式为 Discrete 时有效，采样时间只能是标量。

S-Function Builder 将用户在初始化界面所输入的信息生成 mdlInitializeSizes 回调方法，系统在模型初始化期间调用该方法，用以得到该 S-Function 的基本信息。



4.3.4 数据属性界面(Data Properties)

(1) 输入/输出接口界面(Input/output ports),如图 4.3.5 所示。

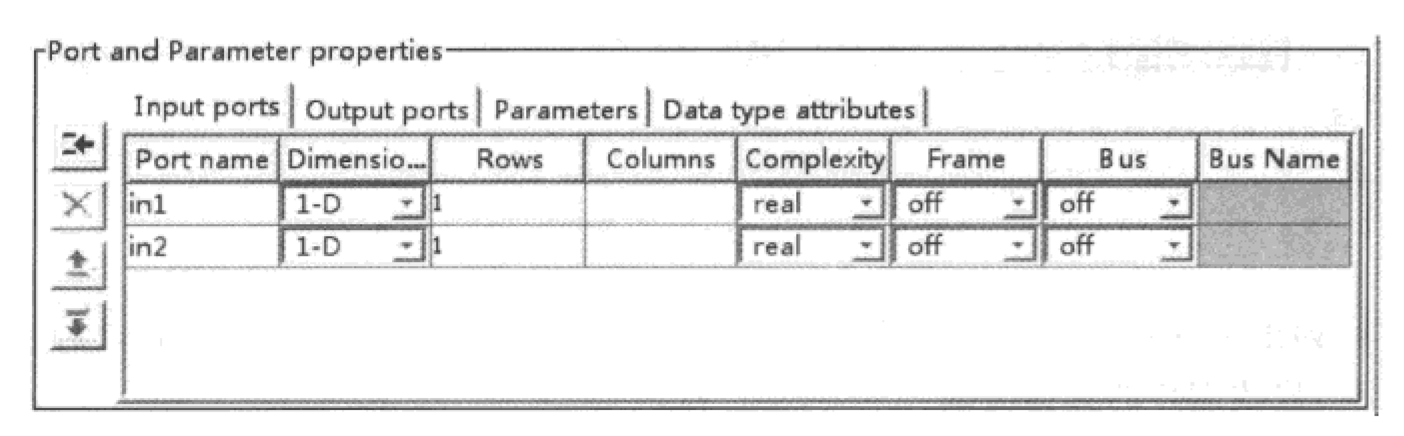


图 4.3.5 输入/输出接口界面

Port and Parameter properties 区域的左侧有 4 个按钮,用来新增、删除、上移、下移相应的端口或参数。

- Port name:指定端口的名称。
- Dimensions:指定端口信号为一维或是多维。选择 1-D 时,允许动态信号维度,这样用户可以不用考虑实际的信号维度。
- Rows:指定信号的行数,输入-1 则表示动态行数。
- Columns:当信号维度为多维时,指定信号的列数。需要注意:如果行数设置为动态,则列数也必须设置为动态或 1。若列数设置为其他数值,虽然编译可以通过,但是由于这个不正确的设置,任何包含该 S-Function 的仿真都将无法执行。
- Complexity:指定该端口支持的信号是实数或是复数。
- Frame:指定该端口是否支持由 Signal Processing Blockset 或 Communications Blockset 生成的基于帧结构的信号。

除了 Port and Parameter properties 区域的选项,用户还可以通过左侧的 Port / Parameter 区域快速定位输入/输出接口以及参数。

(2) 参数界面(Parameters)如图 4.3.6 所示。

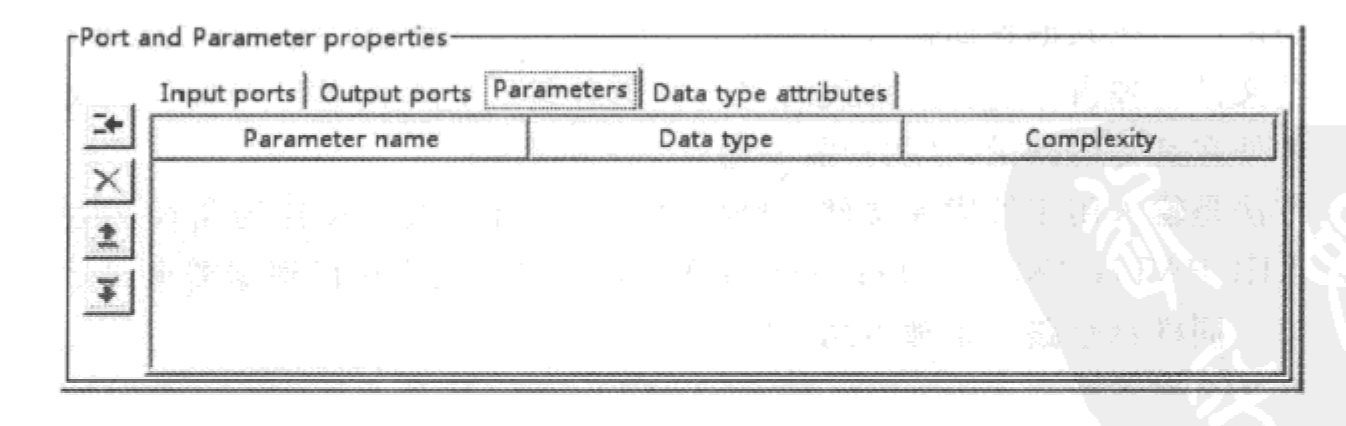


图 4.3.6 参数界面

① Parameter name:参数的名称。

② Data type:指定参数的数据类型。

③ Complexity:指定参数值是实数或是复数。

(3) 数据类型属性界面(Data Type Attributes)如图 4.3.7 所示。

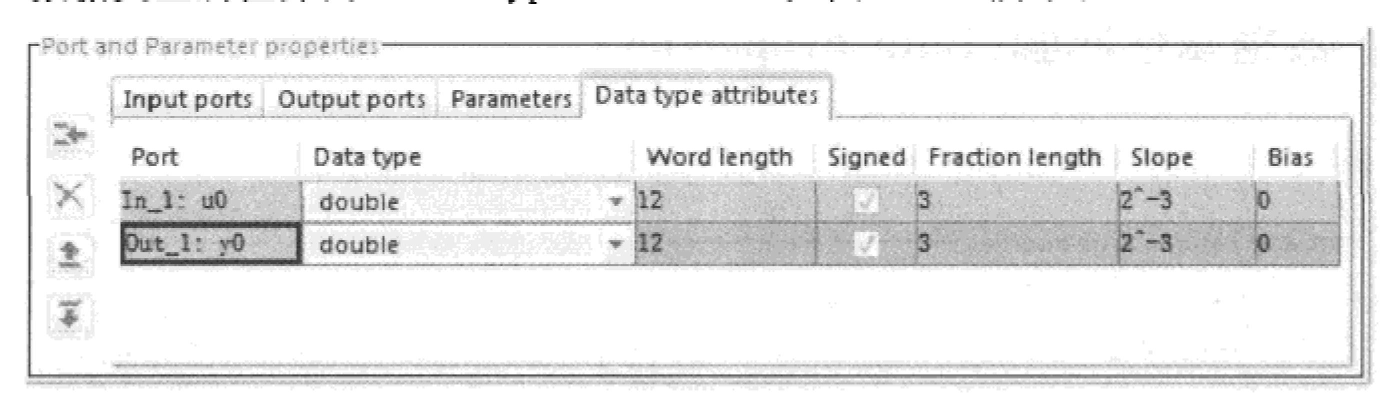


图 4.3.7 数据类型属性界面

该界面列出了 S-Function 所有输入/输出端口的数据类型属性,单击 Data Type 栏右侧的下拉箭头,修改端口的数据类型。

4.3.5 库文件界面(Libraries)

用户可以在这里指定外部代码文件的名称与位置,外部代码是指在其他界面输入的自定义代码中引用到的代码文件,如图 4.3.8 所示。

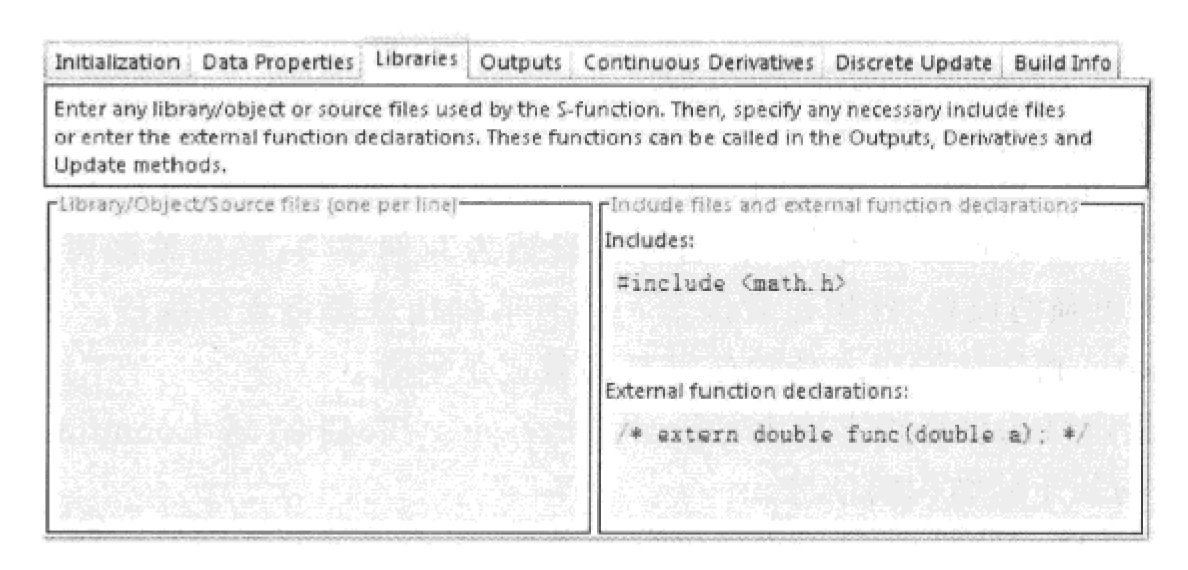


图 4.3.8 库文件界面

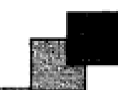
1. Library/Object/Source files 区域

该区域用以声明在其他界面输入的自定义代码中引用到的外部库文件、对象代码以及源文件。每一个文件单独占一行。如果代码文件处在当前工作目录,用户只需声明文件名即可;如果代码文件处在其他的目录,则用户需要输入完整的路径。

另一种方法是输入用于搜索库文件、对象文件、头文件以及源文件的路径,并在路径前分别加上标签 LIB_PATH、INC_PATH、SRC_PATH。当然,用户可以根据需求,尽量多地指定这些文件入口。同样每个路径单独占用一行。

若 MATLAB 安装于 C:\Program Files\MATLAB\,某个 S-Function Builder 文件处于 E:\user\,需要链接的 3 个文件位于以下位置:

```
C:\Program Files\MATLAB\user_object\dx.obj
C:\user_lib\ku.lib
D:\source\yuan.c
```



对应的代码如下：

```
LIB_PATH        $MATLABROOT\user_object
LIB_PATH        C:\user_lib
SRC_PATH  D:\source
dx.obj
ku.lib
yuan.c
```

从上例可以看到，路径标签 LIB_PATH 用来标志对象文件以及库文件路径，SRC_PATH 标志源文件路径，路径与具体文件各自占用一行。$MATLABROOT 表示 MATLAB 的安装路径。如果用户输入了多个 LIB_PATH 路径，则系统根据路径的上下次序，依次搜索。

注意：不要在路径两端加入引号，即使路径名中含有空格，否则编译器将找不到对应的文件。

用户还可以在此输入预处理指令，前缀－D，如－DDEBUG。

2. Includes 区域

该区域用于声明头文件，而这些头文件又声明了用户在其他界面输入的自定义代码中引用到的函数、变量以及宏定义。每个头文件声明占用一行，前缀 #include。

对于标准 C 头文件，需要用尖括弧将文件名括起来，例如 #include<math.h>。

对于用户自定义的头文件，则需要用半角双引号，例如 #include"xxx.h"。

如果用户引用的头文件不处在当前工作目录，则需要在前述 Library/Object/Source files 区域，前缀 INC_PATH，指定该头文件的目录。

3. External function declarations 区域

如果上述 Includes 区域所列出的头文件对一些外部函数未曾声明，则用户必须在此进行声明，每一个函数声明占用一行。S-FunctionBuilder 将这些声明包含在生成的 S-Function 源文件中。这就允许 S-Function 调用这些外部函数，计算函数状态或函数输出。

4.3.6 输出界面(Outputs)

输出界面，如图 4.3.9 所示。

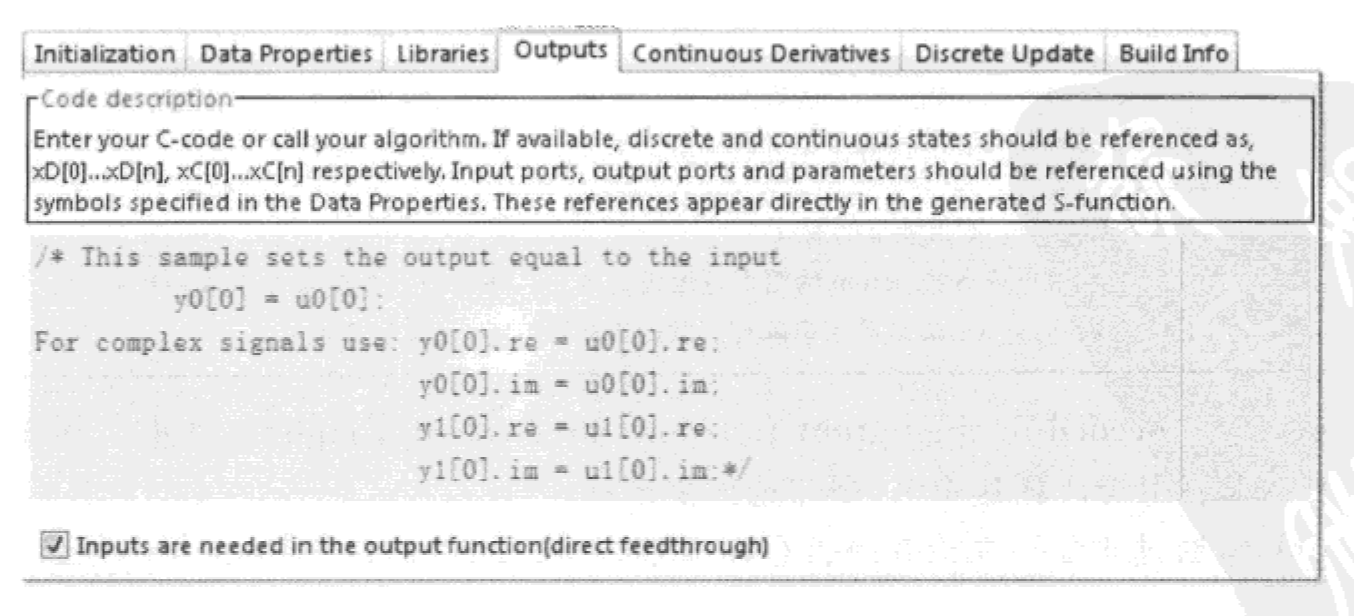


图 4.3.9　输出界面

(1) 用户可以在该区域输入自定义的 C 代码或调用某种算法，代码或算法中的输入/输出接口或参数，必须使用在前述“Data Properties 界面”中定义的符号来表示。系统将在每个采样时间(仿真步长时间或离散 S-Function 的采样时间)计算 S-Function 的输出值。

S-Function Builder 在 mdlOutputs 回调方式中加入了一条调用这个 wrapper 函数的代码。Simulink 在每个采样时间调用 mdlOutputs 方法，mdlOutputs 方法又调用该 wrapper 函数，这时才真正执行用户的代码，并将结果返回 S-Function，作为输出。

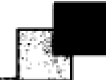
mdlOutputs 方式将把下列某些或全部参数传递给输出 wrapper 函数。

表 4.3.1 mdlOutputs 方式传递参数

参 数	说 明
u0, u1, ... uN	S-Function 输入数组的指针，N 表示在 Input ports 页面中定义的输入端口的数量 在输出 wrapper 函数中出现的参数名 u0……uN 就是在 Input ports 页面中定义的输入端口名。每个数组的宽度即每个输入端口的输入宽度。如果先前指定的输入宽度为－1，则数组的宽度将由 wrapper 函数的 u_width 参数决定(详见表格第 7 行)
y0, y1, ... yN	S-Function 输出数组的指针，N 表示在 Output ports 页面中定义的输出端口的数量 在输出 wrapper 函数中出现的参数名 y0…yN 就是在 Output ports 页面中定义的输出端口名。每个数组的宽度即每个输出端口的输出宽度。如果先前指定的输出宽度为－1，则数组的宽度将由 wrapper 函数的 y_width 参数决定。(详见表格第 8 行) 使用该数组向 Simulink 传递由用户代码计算得到的输出
xD	S-Function 离散状态的数组指针。仅当用户在 Initialization 页面中定义了离散状态，该参数才有效 第一步仿真时，离散状态的初始值由用户在 Initialization 页面中指定。在接下来的采样时间点，离散状态值来自 S-Function 在上一次采样点计算得到的数值(详见下文“Discrete Update 界面”)
xC	S-Function 连续状态的数组指针。仅当用户在 Initialization 页面中定义了连续状态，该参数才有效 第一步仿真时，连续状态的初始值由用户在 Initialization 页面中指定。在接下来的采样时间点，连续状态值来自对上一次采样点状态导数的数值积分(详见下文“Continuous Derivatives 界面”)
param0, p_width0, param1, p_width1, ... paramN, p_widthN	param0, param1, ... paramN 是 S-Function 参数数组的指针，N 表示在 Parameters 页面中定义的参数数量 p_width0, p_width1, ... p_widthN 表示参数数组的宽度。如果某个参数是矩阵，数组矩阵行列数的乘积(即矩阵元素的个数)作为该参数的宽度。例如一个 3 行 2 列矩阵参数的宽度为 6。 仅当用户在 Data Properties 页面中定义了参数，上述 param0……p_widthN 才有效
u_width	S-Function 输入数组的宽度 仅当用户将 S-Function 的输入宽度设置为－1，该参数才会出现在生成的代码中。如果输入是一个矩阵，u_width 则等于矩阵行列数的乘积(即矩阵元素的个数)
y_width	S-Function 输出数组的宽度 仅当用户将 S-Function 的输出宽度设置为－1，该参数才会出现在生成的代码中。如果输出是一个矩阵，y_width 则等于矩阵行列数的乘积(即矩阵元素的个数)

② Inputs are needed in the output function(direct feedthrough)复选框。如果 S-Function 当前的输入值用于计算其输出，则勾选该复选框。

Simulink 将通过该复选框检查是否存在由于直接或间接连接 S-Function 的输入/输出端而形成的代数环。



4.3.7　连续状态求导(Continuous Derivatives)

连续状态求导界面如图 4.3.10 所示。

Initialization | Data Properties | Libraries | Outputs | Continuous Derivatives | Discrete Update | Build Info

Code description

This section is optional and use to calculate the derivatives. It is called only if the S-function has one or more continuous states. The states and derivatives of the S-function are of type double and must be referenced as xC[0], xC[1], etc and dx[0] and dx[1] etc respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```
/*
 * Code example
 *   dx[0] = xC[0];
 */
```

图 4.3.10　连续状态求导界面

如果 S-Function 中含有连续状态,用户可以在这里输入用于求导的代码。连续状态以及导数的数据类型是 double,同时必须用 xC[0]…xC[n]与 dx[0]…dx[n]分别表示。代码中的输入/输出接口及参数,必须使用在“Data Properties 界面”中定义的符号来表示。

生成代码时,S-Function Builder 将用户代码插入一个 wrapper 函数,形式如下,并以 sfun 作为 S-Function 名:

```
void  sfun_Derivatives_wrapper(      constreal_T * u,
constreal_T   * y,
real_T        * dx,
real_T        * xC,
constreal_T   * param0,  /* optional */
int_T         p_width0,       /* optional */
real_T        * param1,  /* optional */
int_T         p_width1,       /* optional */
int_T         y_width,   /* optional */
int_T         u_width)   /* optional */
{
……        /* user code */
}
```

S-Function Builder 在 mdlDerivatives 回调方法中加入了一条调用这个 wrapper 函数的代码。Simulink 在每个采样时间结束时调用 mdlDerivatives 方法,获取连续状态的导数(详见帮助文档“How the Simulink Engine Interacts with C S-Functions”)。Simulink 求解器对导数进行数值积分,以确定下一采样时间的连续状态初值。在接下来的这个采样时间里,Simulink 将更新过的状态回传给 mdlOutputs 方法。

mdlDerivatives 回调方法将如下参数传递给导数 wrapper 函数:

u、y、dx、xC、y_width、u_width、param0、p_width0、param1、p_width1…paramN、p_widthN。

读者可以发现，参数 dx 是一个新的参数，它是一个数组指针，宽度等于用户在“Initialization 界面”指定的连续状态的数量。用户应该使用该数组来返回导数值。

引入这些参数，系统计算导数就如同调用一个带输入/输出与可选参数的函数。用户还可以调用在 Libraries 界面声明了的外部函数。

4.3.8 离散状态更新(Discrete Update)

离散状态更新界面如图 4.3.11 所示。

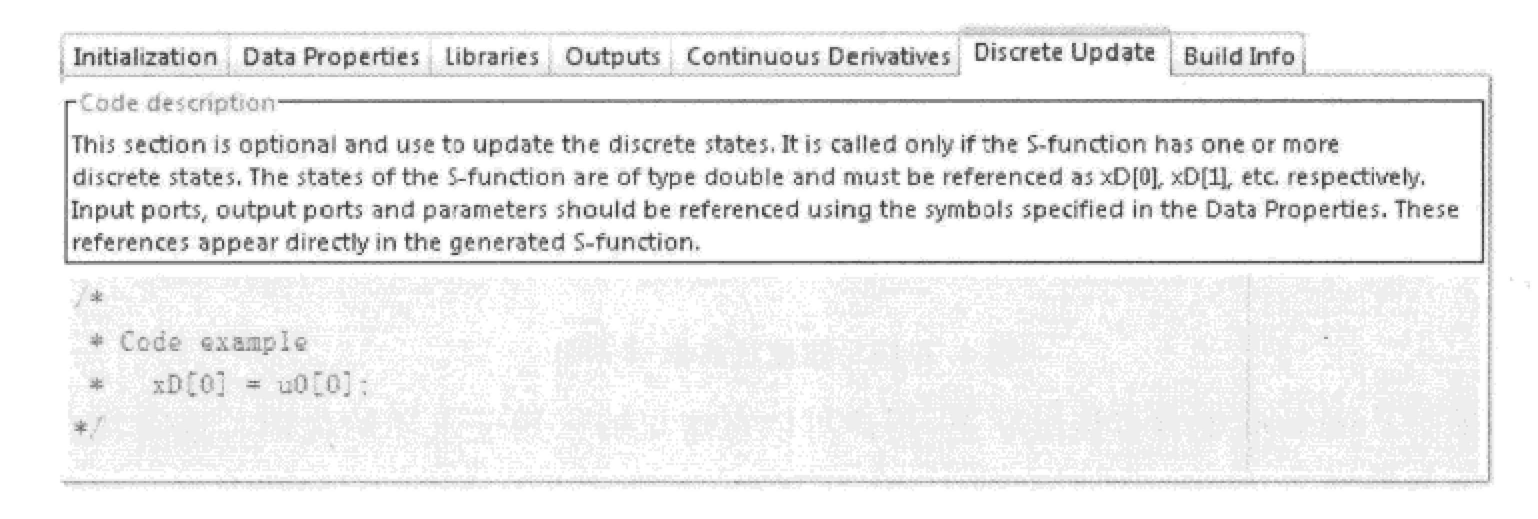


图 4.3.11 离散状态更新界面

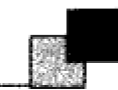
如果 S-Function 中含有离散状态，用户可以在这里输入用于更新离散状态值的代码，在当前采样时间内计算的数值将用于下一采样时间。

离散状态的数据必须用 xD[0]…xD[n]表示。代码中的输入/输出接口及参数，必须使用在“Data Properties 界面”中定义的符号来表示。

生成代码时，S-Function Builder 将用户代码插入一个 wrapper 函数，形式如下，并以 sfun 作为 S-Function 名：

```
void  sfun_Update_wrapper(  constreal_T * u,
constreal_T   * y,
real_T        * xD,
constreal_T   * param0,    /* optional */
int_T         p_width0,        /* optional */
real_T        * param1,    /* optional */
int_T         p_width1,        /* optional */
int_T         y_width,     /* optional */
int_T         u_width)     /* optional */
{
……         /* user code */
}
```

S-Function Builder 在 mdlUpdate 回调方式中加入了一条调用这个 wrapper 函数的代码。Simulink 在每个采样时间结束时调用 mdlUpdate 方法，获取下一采样时间的离散状态值(详见帮助文档“How the Simulink Engine Interacts with C S-Functions”)。在接下来的这个采样时间里 Simulink 将更新过的状态回传给 mdlOutputs 方法。



mdlUpdates 回调方法将 u、y、xD、y_width、u_width、param0、p_width0、param1、p_width1…paramN,p_widthN 等参数传递给更新 wrapper 函数。

用户应使用变量 xD(离散状态)来返回离散状态值。

引入这些参数,系统计算离散状态值就如同调用一个带输入/输出与可选参数的函数。用户还可以调用在 Libraries 界面声明了的外部函数。

4.3.9 编译信息(Build Info)

编译信息界面如图 4.3.12 所示。

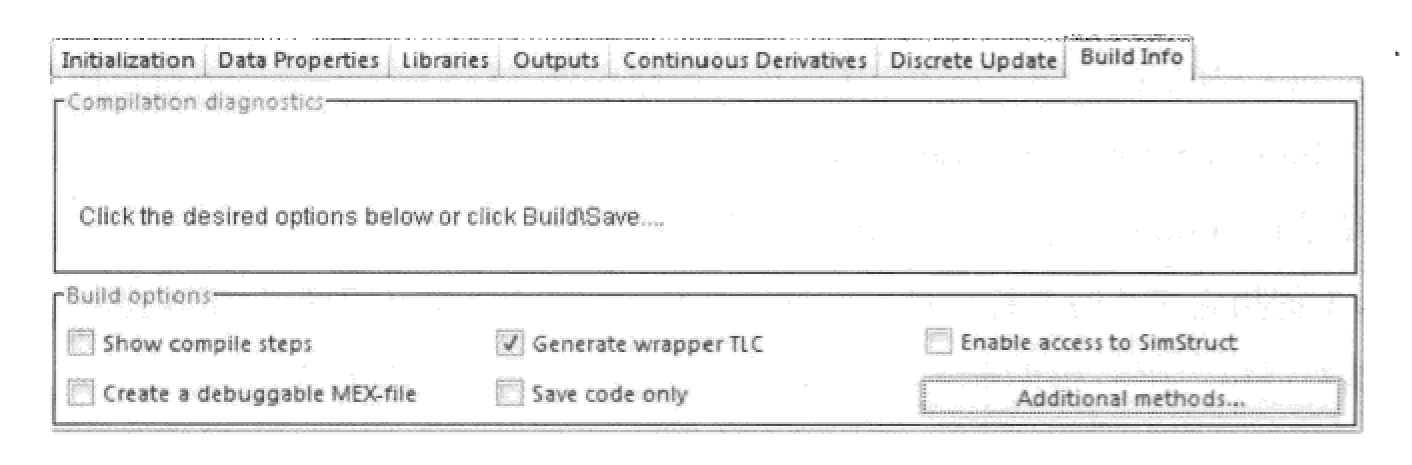


图 4.3.12 编译信息界面

(1) 用户可以在此界面设置一些用于生成 MEX 文件的选项。

① Compilation diagnostics:S-Function Builder 在生成 C 源代码和可执行文件时,显示有关的信息。

② Show compile steps:选中该项,Compilation diagnostics 区域将显示编译的每个步骤。

③ Create a debuggable MEX-file:选中该项,生成的 MEX-file 中将包含调试信息。

④ Generate wrapper TLC:选中该项,将生成 TLC 文件。如果用户不打算让 S-Function 在 Accelerator 模式下运行,也不打算将它用在生成 Real-Time Workshop 代码的模型中,则不需要生成 TLC 文件。

⑤ Save code only:选中该项,将不生成 MEX 文件。

⑥ Enable access to SimStruct:选中该项,允许 S-FunctionBuilder 生成的 wrapper 函数访问 SimStruct (S)。于是用户就可以在 Outputs、Continuous Derivatives、Discrete Updates 等界面中使用 SimStruct 的宏定义及函数。

(2) Additional methods 界面如图 4.3.13 所示。

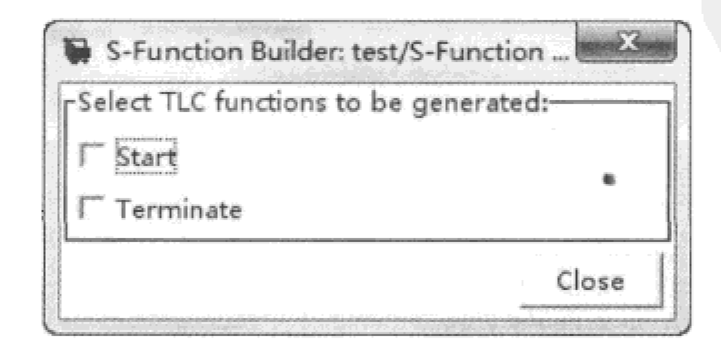


图 4.3.13 附加方法

该对话框允许用户选择在 TLC 文件中增加更多 TLC 方法。

更多信息详见 Real-TimeWorkshop TLC(Target Language Compiler)帮助文档中的 Block Target File Methods 部分。

4.4 创建设备驱动实例

在目前的系统开发过程中,ADC 模块的应用十分广泛,它能够在物理世界与嵌入式研究之间搭起一座桥梁,因此本节以 ADC 模块为例,描述 ADC S 函数的实现过程。

Inlined S-function 不同于 noninlined S-function,前者可直接用于生成实时代码,为此,除了需要编写 noninlined S-function 外,还应该提供相应的 TLC 文件和硬件初始化信息。

编写 Inlinined S-function(内联 S 函数)的步骤如下:

(1) 编写 noninlined C MEX S-function 代码。

注意,不能用它来读写目标硬件的存储器地址,只能通过 TLC 文件才能实现这些操作。

(2) 编写 TLC 文件,用来定制 RTW,为系统模型生成实时 C 代码。

(3) 硬件的初始化方式。

① 定义设备头文件(device. h)。该文件包含有指定 I/O 硬件的宏定义、变量定义、code 库等内容。

② 通过在 TLC 文件的 start 方法,初始化硬件的工作状态。

(4) 创建 ADC S 函数模块。

4.4.1 HC12 模数转换模块

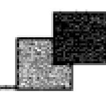
本例将介绍 Freescale HC12 微处理器上的 ADC 驱动模块的创建过程。

1. 编写 nonlined S-function

编写 noninline C MEX S-function 代码,即 ADC_examp. c 源代码如下:

```
#define S_FUNCTION_NAME ADC_examp            /* S 函数名为 ADC_examp */
#define S_FUNCTION_LEVEL 2                   /* 2 级 S 函数 */
/* 定义错误提示信息 */
#define ERR_INVALID_SET_INPUT_DTYPE_CALL \
            "Invalid call to mdlSetInputPortDataType"
#define ERR_INVALID_SET_OUTPUT_DTYPE_CALL \
            "Invalid call to mdlSetOutputPortDataType"
#define ERR_INVALID_DTYPE    "Invalid input or output port data type"

/* 宏定义(需要 simstruct.h 头文件) */
#include "simstruc.h"
#define TRUE     1
```



```
#define FALSE   0
#define N_PAR   5      /* 定义模块的参数个数 */
/*
 * CHANNELARRAY_ARG    - ADC 通道矩阵(0~7 间的一个或多个值),同时定义了信号宽度
 * SAMPLETIME(S)       - 采样时间
 % ATDBANK(S)          - Bank 0 或 Bank 1. 每个 bank 提供 8 通道
 * USE10BITS(S)        - 如果 USE10BITS_ARGC == 1 时, 使用 10 位 ADC ,否则使用 8 位 ADC
 * LEFTJUSTIFY(S)      - 如果 LEFTJUSTIFY_ARGC == 1, 16bit 的结果采用左对齐方式,否则使用右对
 *                       齐(默认)
 */
enum {ATDBANK_ARGC = 0, CHANNELARRAY_ARGC, USE10BITS_ARGC, LEFTJUSTIFY_ARGC, SAMPLETIME_ARGC};
#define ATDBANK(S) (mxGetScalar(ssGetSFcnParam(S,ATDBANK_ARGC)))
#define CHANNELARRAY_ARG(S) (ssGetSFcnParam(S,CHANNELARRAY_ARGC))
#define USE10BITS(S) (mxGetScalar(ssGetSFcnParam(S,USE10BITS_ARGC)))
#define LEFTJUSTIFY(S) (mxGetScalar(ssGetSFcnParam(S,LEFTJUSTIFY_ARGC)))
#define SAMPLETIME(S) (mxGetScalar(ssGetSFcnParam(S,SAMPLETIME_ARGC)))
```

```
/* 模型初始化 */
static void mdlInitializeSizes(SimStruct *S)
{
    const unsigned int *paramPtr = mxGetData( CHANNELARRAY_ARG(S) );
    int nChannels;

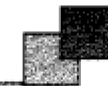
    /* 设置并检测参数个数 */
    ssSetNumSFcnParams(S, N_PAR);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) return;
    ssSetSFcnParamNotTunable(S, 0);
    /* 设置 5 个参数皆为不可调 */
    ssSetSFcnParamNotTunable(S, 1);
    ssSetSFcnParamNotTunable(S, 2);
    ssSetSFcnParamNotTunable(S, 3);
    ssSetSFcnParamNotTunable(S, 4);
    nChannels = mxGetNumberOfElements( CHANNELARRAY_ARG(S) );
    /* 信号输入端口宽度设置为 nChannels */
    if ( !ssSetNumInputPorts( S, 1 ) ) return;
    ssSetInputPortWidth(      S, 0, nChannels );
    /* 信号输出端口宽度设置为 nChannels */
    if ( !ssSetNumOutputPorts( S, 1 ) ) return;
    ssSetOutputPortWidth(      S, 0, nChannels );

    /* 把输入/输出接口的数据类型与用户可选的 8/10 位解析度相关联 */
    if (USE10BITS(S))
    {
      /* 选择 10 位时输入/输出的数据类型定义为 uint16 */
```

```
        ssSetInputPortDataType( S, 0, SS_UINT16 );
        ssSetOutputPortDataType( S, 0, SS_UINT16 );
      } else {
        /* 选择 8 位时输入/输出的数据类型定义为 uint8 */
        ssSetInputPortDataType( S, 0, SS_UINT8 );
        ssSetOutputPortDataType( S, 0, SS_UINT8 );
      }
        ssSetInputPortDirectFeedThrough( S, 0, TRUE );
      /* 采样率个数 */
      ssSetNumSampleTimes( S, 1 );
        /* 选择 */
      ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
      } /* 结束 mdlInitializeSizes 方法 */
```

```
  /* 采样时间初始化 */
  static void mdlInitializeSampleTimes(SimStruct *S)
  {
      ssSetSampleTime( S, 0, SAMPLETIME(S) );
      } /* 结束 mdlInitializeSampleTimes 方法 */
```

```
  /* 模型输出 */
  static void mdlOutputs(SimStruct *S, int_T tid)
  {
      /* 使用特殊类型指针"uPtrs"(本质是一个指针向量)指向 0 号和 1 号输入端口 */
      DTypeId y0DataType; /* SS_UINT8 或 SS_UINT16 */
      int_T y0Width = ssGetOutputPortWidth(S, 0);
      InputPtrsType u0Ptrs = ssGetInputPortSignalPtrs(S,0);
      /* 得到输出端口数据类型 ID,并与输入信号相匹配 */
      y0DataType = ssGetOutputPortDataType(S, 0);
      /* 将输入信号赋给输出端口 */
      switch (y0DataType)
      {
        /* 数据为 uint8 型时的输出 */
        case SS_UINT8:
        {
        uint8_T *pY0 = (uint8_T *)ssGetOutputPortSignal(S,0);
        InputUInt8PtrsType pU0 = (InputUInt8PtrsType)u0Ptrs;
        int i;
        /* 把输入信号赋给输出端口 */
        for( i = 0; i < y0Width; ++i){
        pY0[i] = *pU0[i];
        /* 采用默认的右对齐方式. */
          }
          break;
```



```
        }
        /* 数据为 uint16 型时的输出 */
        case SS_UINT16:
        {
        uint16_T *pY0 = (uint16_T *)ssGetOutputPortSignal(S,0);
        InputUInt16PtrsType pU0 = (InputUInt16PtrsType)u0Ptrs;
          int i;
          for( i = 0; i < y0Width;
          /* 把输入信号赋给输出端口 */
          if (LEFTJUSTIFY(S)) {
          /* 若采用左对齐方式,需左移 6 位. */
          pY0[i] = *pU0[i]<<6;
              } else {
          /* 若采用右对齐方式,不需移位. */
                  pY0[i] = *pU0[i];
              }
          }
          break;
      }
    } /* end switch (y0DataType) */
  } /* 结束 mdlOutputs 方法 */
```

```
/* 该方法执行诸如释放内存等,只有在仿真结束时或 S 函数模块被删除时执行 */
static void mdlTerminate(SimStruct *S)
{
} /* end mdlTerminate */
```

```
#define MDL_START /* #define MDL_START 后,才可调用 mdlStart 方法 */
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{
/* 此方法只在仿真过程中输出一条信息 */
  if (ssGetSimMode(S) == SS_SIMMODE_NORMAL) {
    mexPrintf("\n ADC_examp driver: Simulating initialization\n");
    }
}/* 结束 mdlStart 方法 */
#endif /* MDL_START */
```

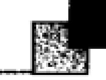
```
/* 判断是否为可接受的数据类型 */
static boolean_T isAcceptableDataType(DTypeId dataType)
{
    boolean_T isAcceptable = (dataType == SS_UINT8 ||
                              dataType == SS_UINT16);
    return isAcceptable;
```

```
}
```

```
#define MDL_SET_INPUT_PORT_DATA_TYPE
/* 设置输入端口数据类型 Function: mdlSetInputPortDataType
=============================================
*    此方法用来动态设置端口的备选数据类型。若选择的数据类型可接受,则调用
     ssSetInputPortDataType 方法设置其类型;否则调用 ssSetErrorStatus
*/
static void mdlSetInputPortDataType(SimStruct *S,
                                    int      port,
                                    DTypeId dataType)
{
    if ( port == 0 ) {
            /* 判断是否接受该数据类型 */
      if( isAcceptableDataType( dataType ) ) {
            /* 把所有数据端口设置为该数据类型 */
            ssSetInputPortDataType( S, 0, dataType );
            ssSetOutputPortDataType( S, 0, dataType );
      } else {
            /* 拒绝该数据类型 */
            ssSetErrorStatus(S,ERR_INVALID_DTYPE);
            goto EXIT_POINT;
      }
    } else {
      /* 只有在已存在的输入端口数据类型未知时调用该方法 */

      ssSetErrorStatus(S, ERR_INVALID_SET_INPUT_DTYPE_CALL);
      goto EXIT_POINT;
    }
EXIT_POINT:
    return;
} /* mdlSetInputPortDataType */
```

```
#define MDL_SET_OUTPUT_PORT_DATA_TYPE
/* 设置输出端口数据类型
=============================================
*    此方法用来动态设置端口的备选数据类型。若选择的数据类型可接受,则调用
     ssSetOutputPortDataType 方法设置其类型;否则调用 ssSetErrorStatus
*/
static void mdlSetOutputPortDataType(SimStruct *S,
                                     int      port,
                                     DTypeId  dataType)
{
    if ( port == 0 ) {
```



```
            /* 判断是否接受该数据类型 */
        if( isAcceptableDataType( dataType ) ) {
            /* 把所有数据端口设置为该数据类型 */
            ssSetInputPortDataType( S, 0, dataType );
            ssSetOutputPortDataType( S, 0, dataType );
        } else {
            /* 拒绝该数据类型 */
            ssSetErrorStatus(S,ERR_INVALID_DTYPE);
            goto EXIT_POINT;
        }
    } else {
        /* 只有在已存在的输出端口数据类型未知时调用该方 */
        ssSetErrorStatus(S, ERR_INVALID_SET_OUTPUT_DTYPE_CALL);
        goto EXIT_POINT;
    }
EXIT_POINT:
    return;
} /* mdlSetOutputPortDataType */
```

```
#define MDL_SET_DEFAULT_PORT_DATA_TYPES
/* 设置默认端口类型
 ======================================
 * 当备选数据类型缺省时,调用该方法为动态类型端口设置数据类型
 */
static void mdlSetDefaultPortDataTypes(SimStruct *S)
{
    /* 将输入端口数据类型设置为 uint8 */
    ssSetInputPortDataType( S, 0, SS_UINT8 );
    ssSetOutputPortDataType( S, 0, SS_UINT8 );
} /* mdlSetDefaultPortDataTypes */
```

```
#define MDL_RTW
static void mdlRTW(SimStruct *S)
{
    uint8_T  atdbank       = (uint8_T) ATDBANK(S);
    uint16_T *channels     = (uint16_T *) mxGetData(CHANNELARRAY_ARG(S));
    uint8_T use10BitRes  = (uint8_T) USE10BITS(S);
    uint8_T leftjustify  = (uint8_T) LEFTJUSTIFY(S);

    /* 向模块写入参数 */
    if (!ssWriteRTWParamSettings(S, 4,SSWRITE_VALUE_DTYPE_NUM,"ATDBank",
                            &atdbank,DTINFO(SS_UINT8, COMPLEX_NO),

                            SSWRITE_VALUE_DTYPE_VECT, "Channels",channels,
```

```
                                mxGetNumberOfElements(CHANNELARRAY_ARG(S)),
                                DTINFO(SS_UINT16, COMPLEX_NO),

                                SSWRITE_VALUE_DTYPE_NUM,"Use10BitRes",
                                &use10BitRes,DTINFO(SS_UINT8, COMPLEX_NO),

                                SSWRITE_VALUE_DTYPE_NUM,"LeftJustify",
                                &leftjustify,DTINFO(SS_UINT8, COMPLEX_NO))) {
            return; /* Simulink 将会报告一个错误 */
        }
    }
```

```
/* ============================================*
 * 确保在无 TLC 文件时不能由 RTW 生成代码
 * ============================================*/
 #ifdef    MATLAB_MEX_FILE   /* 该文件是否编译为 MEX-file */
# include "simulink.c"        /* MEX-file 接口机制 */
#else                    /* 确保在无 TLC 文件时不会被 RTW 使用 */
# error "Attempted use non-inlined S-function ADC_examp.c"
#endif
/* [EOF] ADC_examp.c */
```

(1) mdlInitializeSizes 回调方法：设置 S 函数的状态数量，参数个数，输入口的个数、维度以及直馈标志位，出口的个数、维度，采样率的个数，模块工作参数的大小和模块仿真选项。

① ssSetNumSFcnParams(S, N_PAR)设置参数个数为 5。

② ssSetInputPortWidth(S, 0, nChannels)设置输入口宽度为 nchannels。

③ ssSetOutputPortWidth(S, 0, nChannels) 设置输出口宽度为 nchannels。

④ ssSetInputPortDataType(S, 0, *)设置输入口数据类型为 uint8 或 uint16。

⑤ ssSetOutputPortDataType(S, 0, *) 设置输出口数据类型为 uint8 或 uint16。

⑥ ssSetInputPortDirectFeedThrough(S, 0, TRUE)设置直馈标志位为真。

⑦ ssSetNumSampleTimes(S, 1)设置采样率个数为 1。

(2) mdlInitializeSampleTimes 回调方法：设置采样率、采样时间值和偏移量。

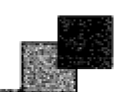
ssSetSampleTime(S,0,SAMPLETIME(S))设置采样时间为 SAMPLETIME(S)。

(3) mdlOutputs 回调方法：在每个仿真步长，计算当前步长的 S 函数输出，并将结果保存在 S 函数输出信号数组。

InputPtrsType u0Ptrs = ssGetInputPortSignalPtrs(S,0)定义了特殊类型指针 uPtrs 指向输入端口。

(4) mdlTerminate 回调方法：执行诸如释放内存等工作，但必须在仿真结束时或 S 函数模块被删除时才能执行的动作。本例中该回调函数为空。

除了上述 S-Function API 外，本代码文件还用到了如下可选 S-Function API：mdlStart，isAcceptableDataType，mdlSetInputPortDataType，mdlSetOutputPortDataType，mdlRTW，mdlSetDefaultPortDataTypes。



其中mdlRTW能够在代码生成过程中评估并将数据类型规范化，在RTW工具生成model.rtw文件时，调用该方法；mdlstart在S-Function中只会被调用一次，其功能为初始化S函数的工作向量、分配存储器地址、设置用户数据、初始化状态等，并且只有用#define定义了MDL_START后，mdlstart才会被调用。

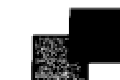
2. 编写TLC文件

```
%implements ADC_examp "C"
%% Function: Start
==========================================================
%%
%%        ADC 初始化代码
%%
%function Start(block, system) Output
/* S-Function "ADC_examp" initialization Block: %<Name> */
%% 设置 ATD Bank 0 或 Bank 1
%assign atdBank = CAST( "Number",SFcnParamSettings.ATDBank)
/* ATDxCTL2 register bits:(查看 HC12 数据手册)
 *  该寄存器用来设置 ADC 的使能位。例如:Bit7 是 ADC 上电位,为 1 时使能。Bit1 是中断
 *  使能位,为 1 时使能。Bit 0 是中断标志位。
 *  更详细功能介绍可查看 HC12 数据手册。
Bit 7    Bit 6    Bit 5    Bit 4    Bit 3    Bit 2    Bit 1    Bit 0
ADPU
AFFC
AWAI
ETRIGLE
ETRGP
ETRGE
ASCIE
ASCIF

 *  本例中该寄存器的值设置如下:
 *  ADPU        1 打开该模块
 *  AFFC        0 在访问结果寄存器之前读取状态寄存器 1(ATDSTAT1),可以正常清除相应 CCF 标志位
 *  AWAI        0 等待模式,A/D 转换继续进行
 *  ETRIGLE     0 外部触发电平/边沿控制位
 *  ETRGP       0 外部触发极性控制位
 *  ETRGE       0 关闭外部触发
 *  ASCIE       0 转换序列完成中断使能位
 *  ASCIF       0 (只读位)
 *  转换为十六进制表示为 0x80
 *  Example: ATD0CTL2 = 0x80;
*/
%% 设置 ATDxCTL2 寄存器的值
```

```
ATD%<atdBank>CTL2 = 0x80;

/* ATD0CTL3 register bits:
 *   该寄存器通常是设置为 0 的,用来控制在后台调试模式时与 A/D 转换相关的处理器行为。
 *   更详细功能介绍可查看 HC12 数据手册。
Bit 7    Bit 6    Bit 5    Bit 4    Bit 3    Bit 2    Bit 1    Bit 0
b7
S8C
S4C
S2C
S1C
FIFO
FRZ1
FRZ0
通过设置 S8C,S4C,S2C,S1C 可以设置转换序列长度:
S8C  S4C  S2C  S1C    转换序列长度
0    0    0    0      8
0    0    0    1      1
0    0    1    0      2
0    0    1    1      3
0    1    0    0      4
0    1    0    1      5
0    1    1    0      6
0    1    1    1      7
1    X    X    X      8
通过设置 FRZ1,FRZ0,可以使能冻结模式的背景调试:
FRZ1     FRZ0       工作状态
0        0          继续转换
0        1          未定义
1        0          完成当前转换,然后暂停
1        1          立即暂停
 *   本例中该寄存器的值设置如下:
 *  b7       0(只读位)
 *  S8C      0 定义转换序列长度
 *  S4C      0 同上一 bit 共同产生作用
 *  S2C      0 同上一 bit 共同产生作用
 *  S1C      0 同上一 bit 共同产生作用
 *  FIFO     0 非先进先出模式
 *  FRZ1     0 设置冻结模式的背景调试使能位
 *  FRZ0     0 同上一 bit 共同产生作用
 *  转化为十六进制表示为 0x00
 *  Example: ATD0CTL3 = 0x00;
 */
```



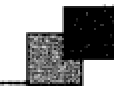
```
%% 设置 ATD0CTL3 寄存器的值
ATD%<atdBank>CTL3 = 0x00;

/* ATD0CTL4 register bits:
 *  该寄存器用来设置 ADC 采样时间和时钟预分频。
 *  更详细功能介绍可查看 HC12 数据手册。
Bit 7    Bit 6    Bit 5    Bit 4    Bit 3    Bit 2    Bit 1    Bit 0
SRES8
SMP1
SMP0
PRS4
PRS3
PRS2
PRS1
PRS0

通过设置 SMP0 和 SMP1 可以设置 4 种采样率：
SMP0  SMP1      采样时间
0      0      2 A/D clock periods
0      1      4 A/D clock periods
1      0      8 A/D clock periods
1      1      16A/D clock periods
通过设置 PRS0-PRS4 可以对 8MHz 的时钟信号进行预分频处理：
PRS0-PRS4            预分频系数
00000
00001                 /4
00010                 /6
00011                 /8
00100                 /10
00101                 /12
00110                 /14
00111                 /16
01xxx
1xxxx
 *  本例中该寄存器的值设置如下：
 *  SRES8  1 ATD 分辨率设置为 8-bits
 *  SMP1   0 采样时间选择
 *  SMP0   0 同上一 bit 共同产生作用
 *  PRS4   0 默认/12 分频
 *  PRS3   0 同上一 bit 共同产生作用
 *  PRS2   1 同上一 bit 共同产生作用
 *  PRS1   0 同上一 bit 共同产生作用
 *  PRS0   1 同上一 bit 共同产生作用
```

```
*   转化为十六进制表示为 0x85
*   Example: ATD0CTL4 = 0x85;
*/
%% 设置 ATD0CTL4 寄存器的值
ATD%<atdBank>CTL4 = 0x85;

* ATD0CTL5 register bits:
*   该寄存器用来设置 A/D 转换模式,通道选择和初始化。若在转换进行过程中对寄存器进行操
*   作,转换进程将会终止并执行写入寄存器的指令。
*   更详细功能介绍可查看 HC12 数据手册。
Bit 7    Bit 6    Bit 5    Bit 4    Bit 3    Bit 2    Bit 1    Bit 0
DJM
DSGN
SCAN
MULT
CD
CC
CB
CA
通过设置 CC,CB,CA 选择输入通道:
CC  CB  CA       输入通道
0   0   0        AN0
0   0   1        AN1
0   1   0        AN2
0   1   1        AN3
1   0   0        AN4
1   0   1        AN5
1   1   0        AN6
1   1   1        AN7
*   本例中该寄存器的值设置如下:
*   DJM     1 结果寄存器数据存储方式为右对齐
*   DSGN    0 以有符号数据形式存储在结果寄存器中
*   SCAN    1 连续转换序列模式
*   MULT    0 使用单通道转换
*   CD      x 通道选择位
*   CC      x 同上
*   CB      x 同上
*   CA      x 同上
*
* 转化为十六进制表示为 0XA0
* Example: ATD0CTL5 = 0xA0
*写入 ATDxCTL5 指定的低 4 位(例如,bits CD,CC,CB,CA)启动一个通道上的 A/D 转换
*/
```



```
%endfunction
%% Function: Outputs
==========================================================
%%
%%  mdlOutputs 方法的代码生成规则
%%
%function Outputs(block, system) Output
/* S-Function "ADC_examp" Block: %<Name> */
%%
%% 确认 ATD bank 为 bank 0 或 bank 1
%assign atdBank = CAST( "Number",SFcnParamSettings.ATDBank)
%%
%assign nPars = SIZE(SFcnParamSettings.Channels,1)
%assign nextChannel = 0
/* 在选定的通道上作 A/D 转换 */
%foreachidx = nPars
%assign channelIdx = CAST("Number",SFcnParamSettings.Channels[idx])
ATD%<atdBank>CTL5 = 0x8%<channelIdx>;
while (CCF%<channelIdx>_%<atdBank>& 0) {
/* 等待转换完成的标志(CCFX) */
}
%assign y = LibBlockOutputSignal(0,"","", %<nextChannel>)
%assign Use10BitResolution = ... CAST("Number",SFcnParamSettings.Use10BitRes)
%%
%% 确认 ATD bank 为 bank 0 或 bank 1
%assign LeftJustify = CAST("Number",SFcnParamSettings.LeftJustify)
%%
%if (%<Use10BitResolution>)
/* 10-bit resolution */
%if (%<LeftJustify>)
/* 左对齐方式 */
%<y>= (uint16_T) ATD%<atdBank>DR%<channelIdx><< 6;
%else
/* 右对齐方式 */
%<y>= (uint16_T) ATD%<atdBank>DR%<channelIdx>;
%endif
%else
/* 8-bit resolution */
%<y>= (uint8_T) ATD%<atdBank>DR%<channelIdx>;
%endif
%assign nextChannel = nextChannel + 1
%endforeach
```

```
% endfunction
```

上述代码并没有以头文件的形式给出设备的存储器状态，而是直接在 Start 方法中定义了 ATD0CTL2 = 0x80，ATD0CTL3 = 0x00，ATD0CTL4 = 0x85，ATD0CTL4 = 0x85。Outputs 中则分别定义了 8 位和 10 位时的输出。

3. 创建 ADC 模块

在 Simulink 库浏览器窗口，选择菜单项 File→New→Library，打开库编辑窗口，并将 Simulink 模块库中的 User-Defined Functions→S-Fuction 模块加入到库编辑窗口中，如图 4.4.1 所示。

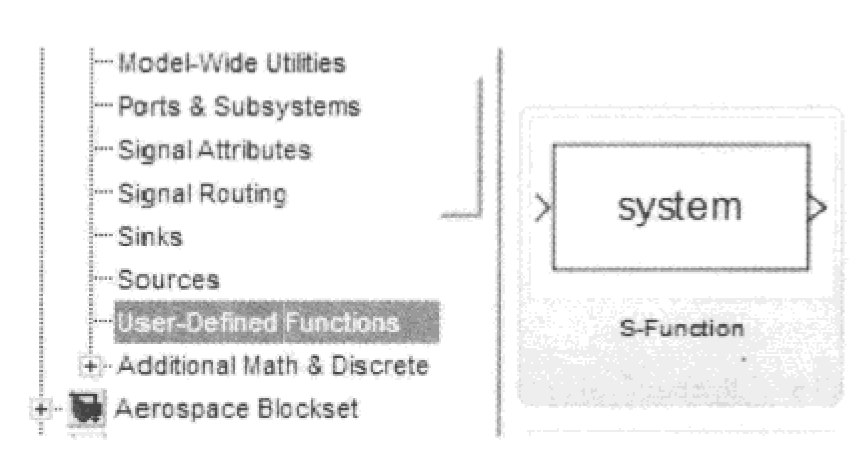


图 4.4.1　S 函数模块

双击 S-Function 模块，设置其对应的 S 函数名为 ADC_examp，如图 4.4.2 所示。并命名为 ADC_examp_lib.mdl，保存于同一个目录下，如图 4.4.3 所示。

为了使用方便，对该模块作 Mask 处理，以便能够用 GUI 窗口设定模块参数。右击模块，在弹出右键菜单中选择 Mask S-Function 命令，如图 4.4.4 所示。

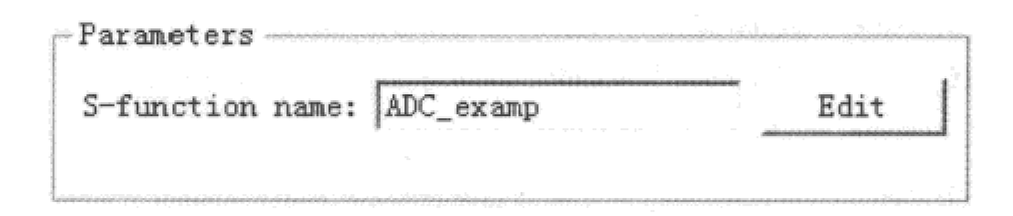


图 4.4.2　设置 S 函数名

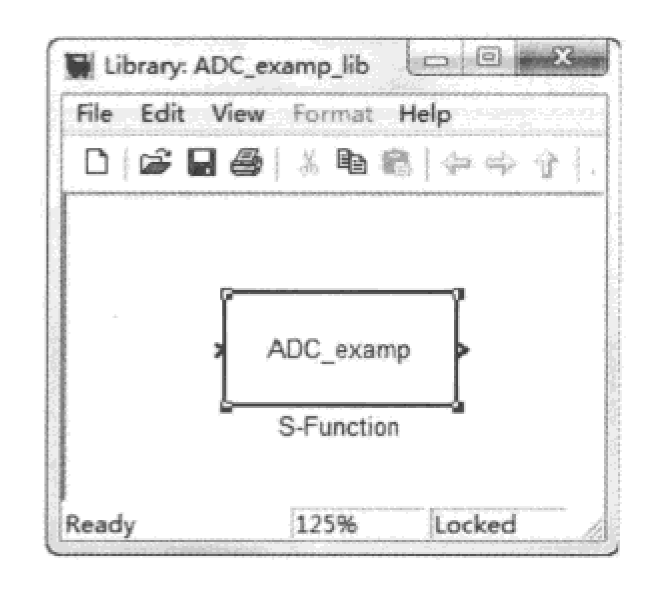


图 4.4.3　S 函数模型

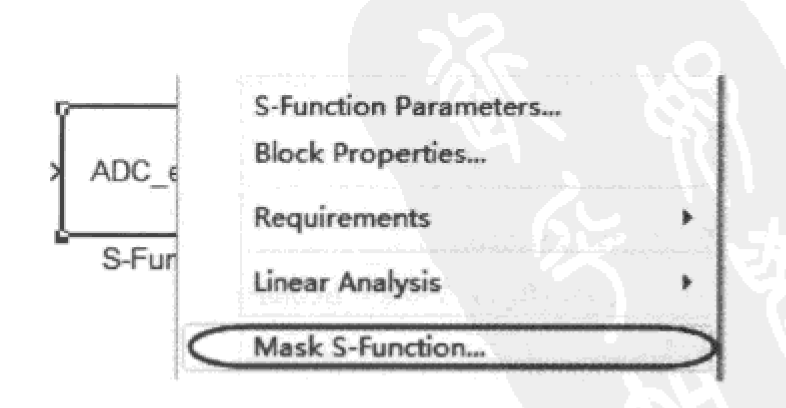
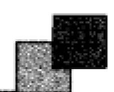


图 4.4.4　封装 S 函数



在 Mask Editor 中选择 parameters 选项卡，单击按钮添加 5 个参数，如图 4.4.5 所示。

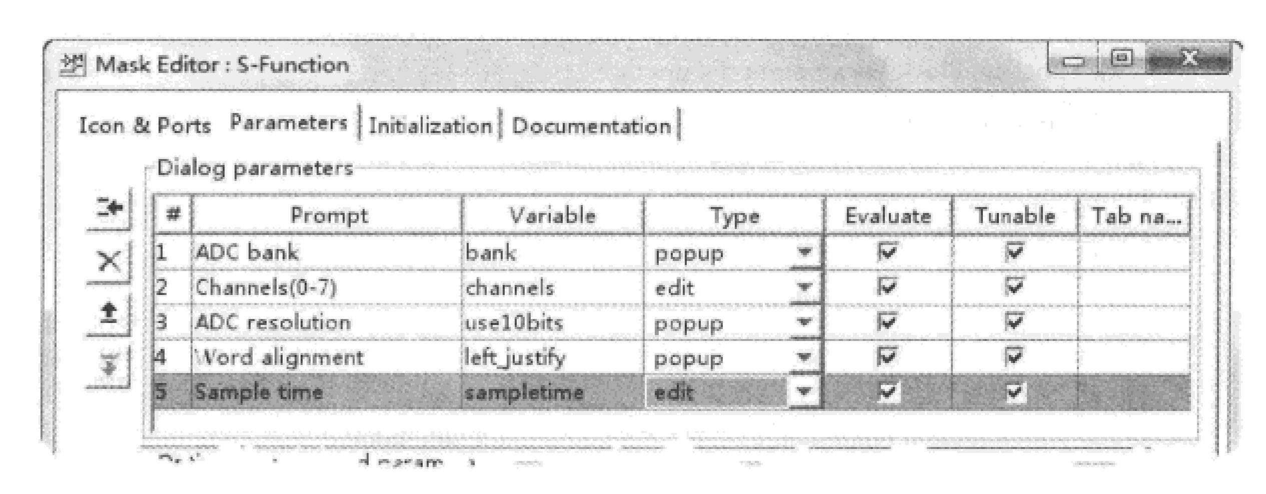
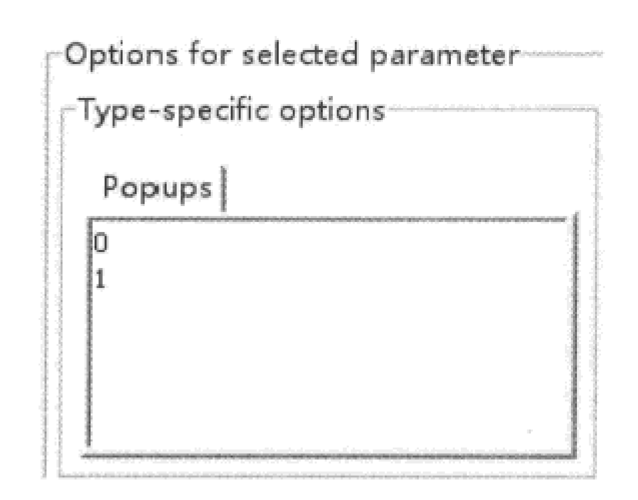


图 4.4.5　添加参数

其中 Type 选项中，popup 表示下拉菜单型参数，edit 表示用户输入型参数。因此，需要在在 Options for selected parameters 区域为下拉菜单型参数添加菜单选项。

首先选中 ADC bank 参数，设置如图 4.4.6 所示。

然后选中 ADC resolution 参数，设置如图 4.4.7 所示。



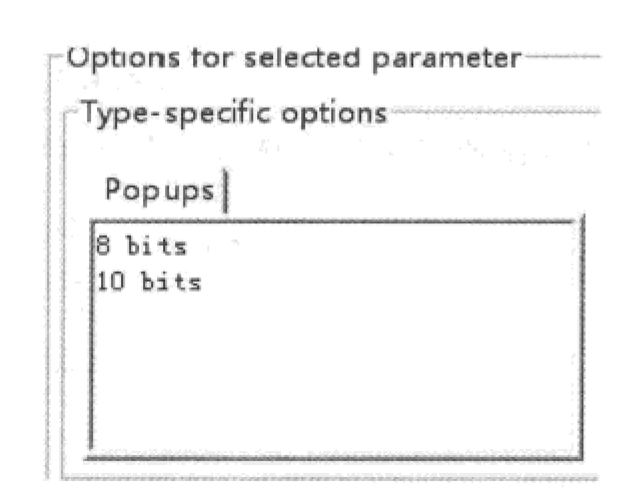


图 4.4.6　ADC bank 参数　　　　图 4.4.7　ADC resolution 参数

再选中 Word alignment 参数，设置如图 4.4.8 所示。

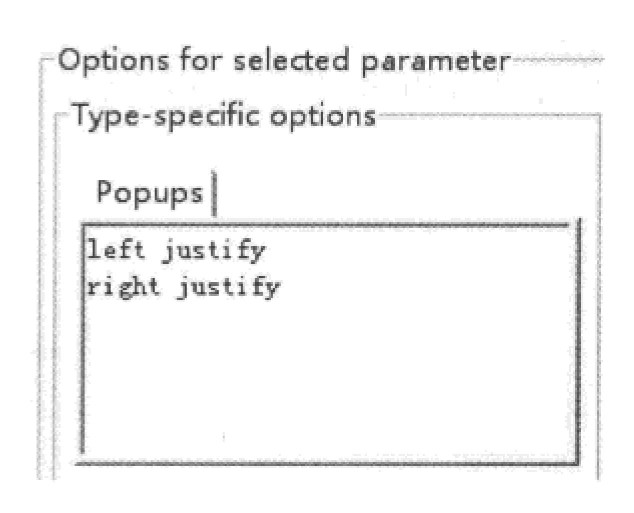


图 4.4.8　ADC resolution 参数

完成设置后单击 apply 按钮，这时再次双击模块时，将会弹出一个参数设置对话框要求用户输入参数，如图 4.4.9 所示。

在 Mask Editor 中设置好参数后，还需要将这些参数与已编写的 S-Function 参数关联起

来，用户可以在 S-Function Parameters 中完成该任务。

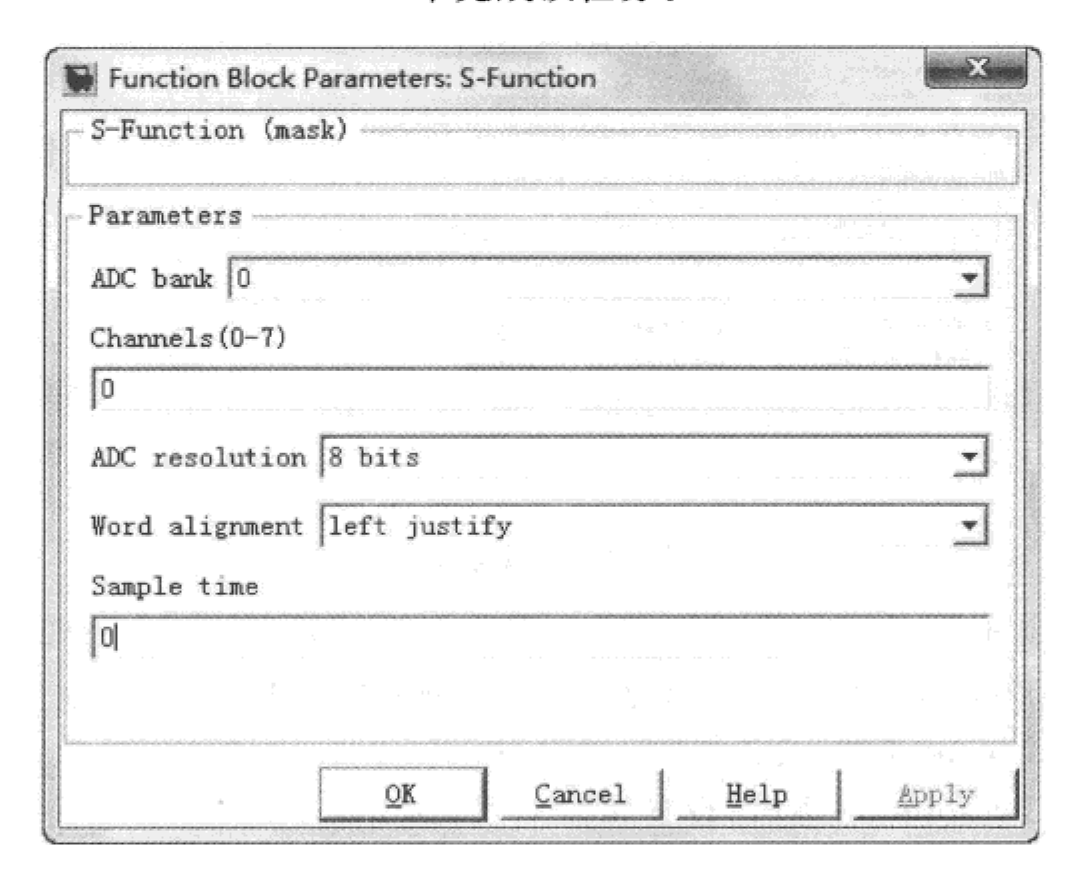


图 4.4.9　参数设置对话框

右击模块，在弹出菜单中选择 S-Function Parameters，图 4.4.10 所示。

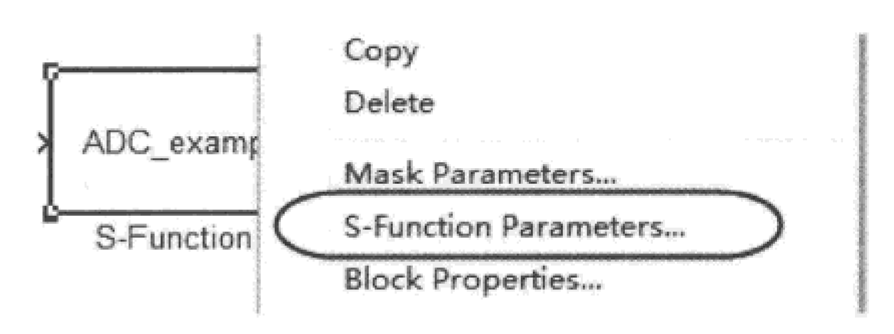


图 4.4.10　S 函数参数菜单

在 parameters 区域的 S-Fuction Parameters 中输入：uint8(bank-1)，uint16(channels)，uint8(use10bits-1)，uint8(left_justify-1)，sampletime，如图 4.4.11 所示。应当注意的是参数的数据类型应该与已编写的 S-Function 中声明的参数相同，并且排列顺序应与 Mask Editor 中定义的参数顺序相一致。

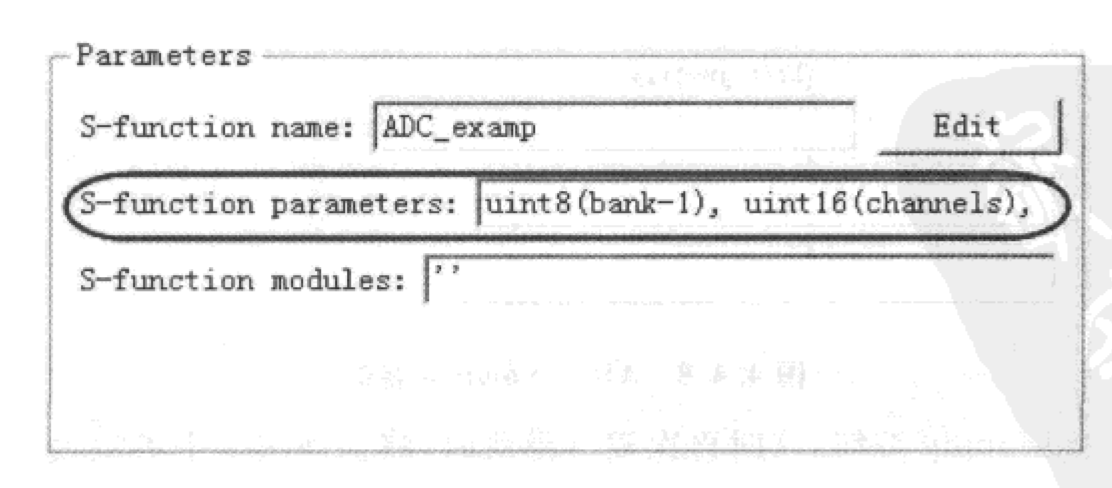
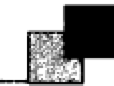


图 4.4.11　设置 S 函数参数



保存模型后，将 ADC_examp. c 保存在当前工作目录下，并在命令行输入以下代码，编译该 S 函数，即可完成该 Inlined Deviec Driver。

```
mex ADC_examp.c
```

4. 测试模型

若缺乏硬件，这样的模型是无法仿真的，读者也无法得知 S 函数是否正确。这里设计一个简单的模型，并对原 S 函数稍作修改，利用参数传递，部分验证 S 函数的正确性。

原始的 S 函数代码：

```
switch (y0DataType)
    {
      /* 数据为 uint8 型时的输出 */
      case SS_UINT8:
      {
        uint8_T *pY0 = (uint8_T *)ssGetOutputPortSignal(S,0);
        InputUInt8PtrsType pU0 = (InputUInt8PtrsType)u0Ptrs;
        int i;
        /* 把输入信号赋给输出端口 */
        for( i = 0; i < y0Width; ++i){
        pY0[i] = *pU0[i];
        /*采用默认的右对齐方式. */
             }
             break;
```

调整如下：

```
switch (y0DataType)
    {
        /* 数据为 uint8 型时的输出 */
        case SS_UINT8:
        {
          uint8_T *pY0 = (uint8_T *)ssGetOutputPortSignal(S,0);
          InputUInt8PtrsType pU0 = (InputUInt8PtrsType)u0Ptrs;
          int    i;
          /* 把输入信号赋给输出端口 */
          for( i = 0; i < y0Width; ++i){
          pY0[0] = N_PAR;
          /*采用默认的右对齐方式. */
               }
               break;
        y[1] = SAMPLE_TIME ;     //输出的第 2 个元素为参数:采样时间
}
```

再次编译并进行仿真，可以看到 S 函数模块正确输出了预先定义的参数个数，如图4.4.12所示。

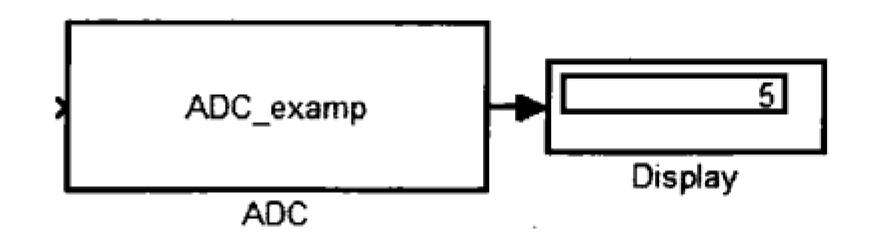


图 4.4.12　S 函数模块仿真结果

4.4.2　DAS1600 数据输入模块

本节以 DAS1600 系列数据采集卡为例，编写其数字输入(digital input)模块。全过程包括编写 S 函数、TLC 文件，以及模块封装与简单测试。由于涉及硬件，若用户手头并无实际板卡，可不必拘泥于函数中各种赋值、判断、输出等算法，仅需了解实际 S 函数的结构。若要深入研究，请参考 DAS1600 系列数据采集卡的用户手册。

1. 编写 nonlined S-function

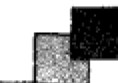
首先根据 4.2.2 节中的 S 函数模板，编写非内联的 S 函数(nonlined S-function)，为与上文保持一致，这里依然分段说明 S 函数。

(1) 文件头部。

```
//定义函数名，函数级别，加入头文件
#define S_FUNCTION_NAME das16di
#define S_FUNCTION_LEVEL 2
#include <stdlib.h>      /* malloc(), free(), strtoul() */
#include "simstruc.h"
```

```
//该 C 文件仅用于生成 C-MEX S 函数
#ifndef MATLAB_MEX_FILE
#error "das16di.c can only be used as a C-MEX S-Function."
#endif
```

```
//定义各种变量
#define NUM_PARAMS              4          //定义参数个数为 4
#define BASE_ADDRESS_PARAM      (ssGetSFcnParam(S,0))
                                           //获取第 1 个参数数组：基地址
#define NUM_CHANNELS_PARAM(ssGetSFcnParam(S,1))
                                           //获取第 2 个参数数组：通道数
#define SAMPLE_TIME_PARAM       (ssGetSFcnParam(S,2))
                                           //获取第 3 个参数数组：采样时间
#define ACCESS_HW_PARAM         (ssGetSFcnParam(S,3))
                                           //获取第 4 个参数数组：硬件访问使能
#define NUM_CHANNELS          ((uint_T) mxGetPr(NUM_CHANNELS_PARAM)[0])
                        //获取通道数参数数组的第 1 个元素，并转换为无符号整型
#define SAMPLE_TIME           ((real_T) mxGetPr(SAMPLE_TIME_PARAM)[0])
                        //获取采样时间参数数组的第 1 个元素，并转换为实型
```

```
#include "das16di.h"        //加入硬件头文件
#ifndef ACCESS_HW
#define ACCESS_HW           (mxGetPr(ACCESS_HW_PARAM)[0] != 0.0)
#endif                      //获取硬件访问使能参数数组的第 1 个元素
#define BASE_ADDR_PARAM_STRLEN      128      //定义基地址参数数组的长度
```

```
typedef struct {            //定义结构体 DIInfo,其中包含变量 baseAddr
    uint_T     baseAddr;
} DIInfo;
```

(2) 参数检查与错误处理。正如第 4.2.2 节提到的,对于需要引入参数的 S 函数,一般都要进行参数检查,在模型初始化时应首先调用该回调方法。

```
#define MDL_CHECK_PARAMETERS
#if defined(MDL_CHECK_PARAMETERS) && defined(MATLAB_MEX_FILE)
```

```
static void mdlCheckParameters(SimStruct *S)
{  static char_T errMsg[256];  //定义一个静态变量,用于保存错误信息文字
   boolean_T allParamsOK = 1;  //所有参数正确标志位
//判断地址是否为字符类型
   if (!mxIsChar(BASE_ADDRESS_PARAM))
   {  sprintf(errMsg, "Base address parameter must be a string.\n");
      allParamsOK = 0;
      goto EXIT_POINT;
   }
```

```
//判断通道数量是否为标量数值,如果为一组向量,则退出
   if (mxGetNumberOfElements(NUM_CHANNELS_PARAM) != 1)
   {  sprintf(errMsg, "Number of channels parameter must be a scalar.\n");
                                        //将错误信息文字保存在静态变量 errMsg
      allParamsOK = 0;
      goto EXIT_POINT;
   }
//调用 diIsNumChannelsParamOK 判断通道数是否在预定的范围,该函数定义在 das16di.h
   if (!diIsNumChannelsParamOK(NUM_CHANNELS))
   {  sprintf(errMsg, "The number of channels must be between 1 and %d\n",DI_MAX_CHANNELS);
//将错误信息文字保存在静态变量 errMsg
      allParamsOK = 0;
      goto EXIT_POINT;
   }
```

```
//判断采样时间是否为标量数值,
   if (mxGetNumberOfElements(SAMPLE_TIME_PARAM) != 1)
   {  sprintf(errMsg, "Sample Time must be a positive scalar.\n");
      allParamsOK = 0;
```

```
            goto EXIT_POINT;
        }
```

```
    //判断硬件访问标志位,或为 0,或为 1,如果为其他数字则退出
        if (ACCESS_HW != 0 && ACCESS_HW != 1) {
            sprintf(errMsg, "Hardware access parameter is %d, expecting 0 or 1.\n", ACCESS_HW);
    //将错误信息文字保存在静态变量 errMsg
            allParamsOK = 0;
            goto EXIT_POINT;
        }
```

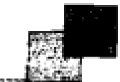
```
    //错误处理
    EXIT_POINT:
        if (!allParamsOK)
        ssSetErrorStatus(S, errMsg);     //将变量 errMsg 的错误信息输出
    }
    #endif
```

(3) 模型初始化。这是 Simulink 引擎调用的第一个回调方法,首先进行参数检查,确保正确后再定义采样时间个数、输入/输出口个数、输出口维度等。

```
static void mdlInitializeSizes(SimStruct * S)
{   ssSetNumSFcnParams(S, NUM_PARAMS);
    if (ssGetNumSFcnParams(S) == ssGetSFcnParamsCount(S))
        {   mdlCheckParameters(S);   //如果参数数目正确,则开始检查参数数值
            if (ssGetErrorStatus(S) != NULL)
                return;              //如果参数值不正确,终止仿真
        }
    else
            return;                  //如果参数数目不正确,终止仿真
//设置所有的参数,在仿真阶段不可调
    {   int_T i;
        for (i = 0; i < NUM_PARAMS; i++)
            ssSetSFcnParamNotTunable(S, i);
    }
    ssSetNumSampleTimes( S, 1);        //设置采样时间个数为 1
    ssSetNumInputPorts( S, 0);         //源模块,因此设置模块输入口数量为 0
    ssSetNumOutputPorts( S, 1);        //设置输出口数量为 1
    ssSetOutputPortWidth(S, 0, NUM_CHANNELS);//设置输出口维度为通道数
}
```

(4) 初始化采样时间。根据输入参数,定义采样时间值。

```
static void mdlInitializeSampleTimes(SimStruct * S)
{   ssSetSampleTime(S, 0, SAMPLE_TIME);   //根据输入参数设置采样时间
    ssSetOffsetTime(S, 0, 0.0);           //设置采样时间偏移量为 0
```



```
}
```

(5) 回调方法 mdlStart。判断硬件访问的使能位,如果允许访问,则继续判断硬件是否准备就绪,否则即认为硬件无法访问,仿真终止。如果用户没有实际板件,应定义使能位为 0,不访问硬件。

```
#define MDL_START
#if defined(MDL_START)
static void mdlStart(SimStruct *S)
{  if (ACCESS_HW)
   {  //初始化结构体 diInfo
      DIInfo *diInfo = ssGetUserData(S);
      char_T baseAddrStr[BASE_ADDR_PARAM_STRLEN];
      uint_T baseAddr;
      mxGetString(BASE_ADDRESS_PARAM, baseAddrStr, BASE_ADDR_PARAM_STRLEN);
      baseAddr = (uint_T) strtoul(baseAddrStr, NULL, 0);
      if (diInfo != NULL)
          free(diInfo);
      if ((diInfo = malloc(sizeof(DIInfo))) == NULL)
          {  ssSetErrorStatus(S,"Memory Allocation Error\n");
             return;
          }
      diInfo->baseAddr = baseAddr;
      ssSetUserData(S, (void*) diInfo);
      mexPrintf("\ndas16di: Hardware Access Enabled\n");
   }
      else if (ssGetSimMode(S) == SS_SIMMODE_NORMAL)
      mexPrintf("\ndas16di: Hardware Access Disabled\n");
}
#endif
```

(6) 计算模块输出。当硬件可访问时,调用函数 diGetInputOnChannel 计算输出,该函数定义在头文件 das16di. h;若硬件不可访问,各通道均输出 0。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{  real_T *y = ssGetOutputPortRealSignal(S,0);
   uint_T i;
   if (ACCESS_HW)              //如果硬件可访问
   { DIInfo *diInfo = ssGetUserData(S);
      uint_T baseAddr = diInfo->baseAddr;
      uint_T inputs = diReadAllInpChannels(baseAddr);
      for (i = 0; i < NUM_CHANNELS; i++)
      {y[i] = diGetInputOnChannel(i,inputs);} //将各通道数据依次输出
   }
   Else                          //如果硬件不可访问
```

```
    {for (i = 0; i < NUM_CHANNELS; i + + )
        {y[i] = 0.0;}        //各通道输出为 0
    }
}
```

(7) 函数结束。

```
static void mdlTerminate(SimStruct * S)
{  DIInfo * adcInfo = ssGetUserData(S);
   free(adcInfo);
   ssSetUserData(S,NULL);
}
```

(8) 回调方法 mdlRTW。该方法主要用于 RTW 代码生成工具生成 model. rtw 文件。

```
# define MDL_RTW
# if defined(MDL_RTW) && (defined(MATLAB_MEX_FILE) || defined(NRT))

static void mdlRTW(SimStruct * S)
{  char_T baseAddrStr[BASE_ADDR_PARAM_STRLEN];
   mxGetString(BASE_ADDRESS_PARAM, baseAddrStr,
                                         BASE_ADDR_PARAM_STRLEN);
   (void)ssWriteRTWParamSettings(S, 1, SSWRITE_VALUE_QSTR,
                                         "BaseAddress",baseAddrStr);
}
# endif
```

(9) 文件尾部。由于该 C 文件仅用于生成 C-MEX S 函数,因此这里只需包含 simulink. c 文件。

```
# include "simulink.c"      /* Required include for MEX interface mechanism */
```

2. 编写 TLC 文件

```
% implements "das16di" "C"
```

```
% % 函数 BlockInstanceSetup
% function BlockInstanceSetup(block, system) void
  % if !EXISTS("Drt_das16di")
    % assign ::Drt_das16di = 1
    % openfile buffer
    # include "das16di.h"
    % closefile buffer
    % <LibCacheIncludes(buffer)>
  % endif
% endfunction % % BlockInstanceSetup
```

```
% % 函数 Outputs
```

```
%function Outputs(block, system) Output
  /*  %<Type> Block: %<Name> ( %<ParamSettings.FunctionName>) */
  {
    %assign numChannels = LibDataOutputPortWidth(0)
    %assign baseAddr = SFcnParamSettings.BaseAddress
   uint_T diInput = diReadAllInpChannels( %<baseAddr>);
    %assign rollVars = ["Y"]
    %assign rollRegion = [0: %<numChannels-1>]
    %roll idx = rollRegion, lcv = RollThreshold, block, "Roller", rollVars
    %assign y = LibBlockOutputSignal(0, "", lcv, idx)
      %<y> = diGetInputOnChannel( %<idx>, diInput);
      %endroll
  }
%endfunction %% Outputs
```

3. 头文件

涉及硬件的头文件，这里仅简单罗列，不作解释。

● das16di.h

```
#ifndef __DAS16DI__
#define __DAS16DI__
#include "tmwtypes.h"
#define DI_MAX_CHANNELS      (4)        //定义通道数
#define diIsNumChannelsParamOK(n) (1 <= n && n <= DI_MAX_CHANNELS)
                                        //判断 S 函数参数是否与 I/O 板卡兼容
#include "drt_comp.h"
#define DIGITAL_IO_REG(bA) (bA + 0x3)         //硬件寄存器
#define diReadAllInpChannels(bA)        //访问 I/O 板卡上的数字输入部分
(ReadByteFromHwPort(DIGITAL_IO_REG(bA))&0xf)
#define diGetInputOnChannel(i,v)        ((real_T) ((v >> i) & 0x1))
#endif /*  __DAS16DI__  */
```

● drt_comp.h

```
#ifndef __DRT_COMPILERS__
#  define __DRT_COMPILERS__
#ifdef __WATCOMC__
#  include <dos.h>
#  include <conio.h>
#  define EnableInterrupts              _enable()
#  define DisableInterrupts             _disable()
#  define GetIntrVector(num)            _dos_getvect(num)
#  define SetIntrVector(num,isr)        _dos_setvect(num,isr)
#  define ISR_PTR_TYPE                  _interrupt _far
```

```
# define ReadByteFromHwPort(addr)        inp(addr)
# define WriteByteToHwPort(addr,val)     outp(addr,val)
#else
# ifdef MATLAB_MEX_FILE
#   define ACCESS_HW                  0
#   define ReadByteFromHwPort(addr)        0
#   define WriteByteToHwPort(addr,val)   /* do nothing */
#  endif
# endif /* _WATCOMC_ */
#endif /* _DRT_COMPILERS_ */
```

4. 设置 S 函数模块

由于该 S 函数含有参数检查，为避免因参数检查失败而无法继续其他操作，首先进行 S 函数的封装，然后再指定 S 函数名。

(1) 模块封装。右击 S-Function 模块，选择菜单项 Mask S-Function…，打开封装对话框。

① Icon & Ports 面板。输入以下代码，定义模块的显示文字与端口文字，如图 4.4.13 所示。

Digital Input di

S-Function

图 4.4.13 S 函数外观文字

```
disp('Digital Input');
port_label('output', 1, 'di');
```

② Parameters 界面。新建 4 个参数，如图 4.4.14 所示，其中 Prompt 列表示封装之后，模块对话框各条目的提示文字，Variable 列表示该参数所对应的变量名。

Icon & Ports | Parameters | Initialization | Documentation

Dialog parameters

#	Prompt	Variable	Type	Evaluate	Tunable	Tab name
1	Base Address	ba	edit	☐	☐	
2	Channels	ch	edit	☑	☐	
3	Sample Time	st	edit	☑	☐	
4	Access Hardware	ah	edit	☑	☐	

图 4.4.14 Parameters 界面

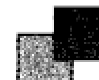
基地址参数 Base Address 的输入值为字符，因此必须取消该参数的 Evaluate 项目，否则将来仿真时系统会提示无法处理该类数据。

③ Initialization 界面。在初始化面板，使用函数 set_param( )，预定义 S 函数模块的参数，例如：

```
set_param(gcb,'ba','0x300','ch','2', 'st','0.1','ah','0')
```

用户还可以在 Documention 面板输入模块更详细的描述文字，本文从略。封装完成后，双击 S 函数模块，得到封装后的模块对话框，如图 4.4.15 所示。

(2) 模块定义。右击 S-Function 模块，选择菜单项 Look Under Mask，打开 S 函数参数设



置对话框，定义其函数名与变量名，如图 4.4.16 所示。

变量名应与 Parameters 面板中定义的一致。

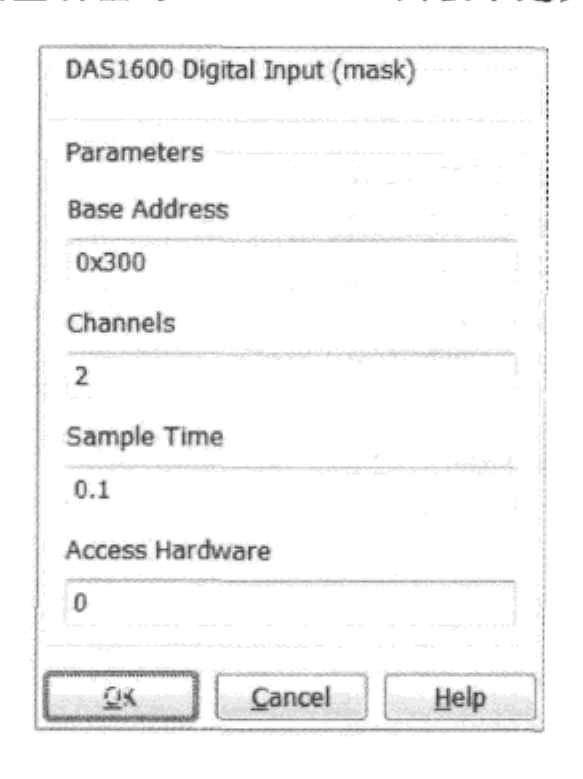


图 4.4.15 封装后的模块对话框

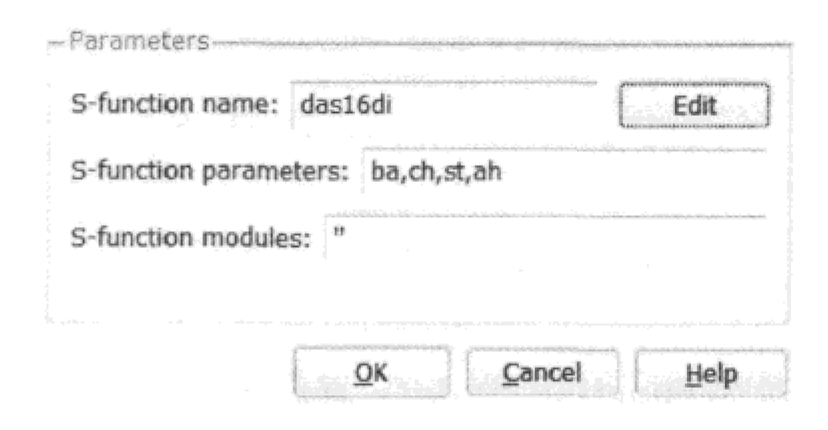


图 4.4.16 S 函数参数设置对话框

确定后，得到最终的模块，如图 4.4.17 所示。

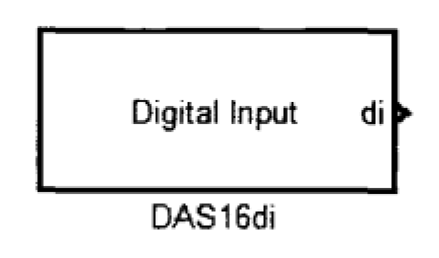


图 4.4.17 DAS1600 数据输入模块

5. 测试模型

若缺乏硬件，这样的模型是无法仿真的，读者也无法得知 S 函数是否正确。这里设计一个简单的模型，并对原 S 函数稍作修改，利用参数传递，部分验证 S 函数的正确性。

原始的 S 函数代码：

```
static void mdlOutputs(SimStruct * S, int_T tid)
...
    else
    {for (i = 0; i < NUM_CHANNELS; i + + )
        {y[i] = 0.0;}
    }
```

调整为：

```
static void mdlOutputs(SimStruct * S, int_T tid)
...
    else
    {   y[0] = NUM_CHANNELS;        //输出的第 1 个元素为参数:通道数
        y[1] = SAMPLE_TIME ;        //输出的第 2 个元素为参数:采样时间
```



```
}
```

再次编译,并根据图 4.4.15 所设定的参数进行仿真,可以看到 S 函数模块正确输出了预先定义的通道数与采样时间,如图 4.4.18 所示。

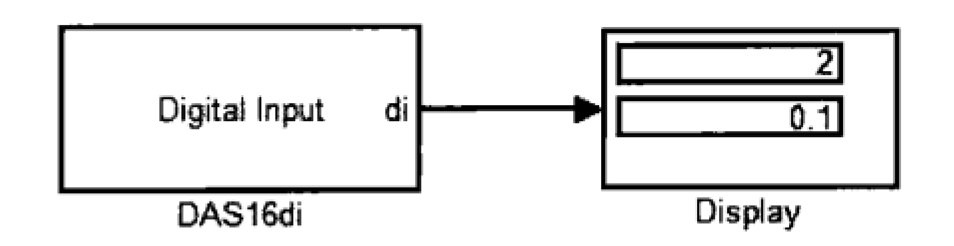


图 4.4.18 S 函数模块仿真结果

4.4.3 S-Function Builder

1. 创建 S-Function Builder 驱动模型

下面将使用 S-Function Builder 继承一个实现两数相加功能 C 代码。将代码命名为 add.c 并保存在当前目录下,代码如下:

```
double add(double a,double b)
{   double sum = 0;
    sum = a + b;
    return sum;
}
```

按照图 4.4.19 新建一个包含 S-Function Builder 的 Simulink 模型。Add 模块完成相同的两数相加功能,用于验证结果的正确性。

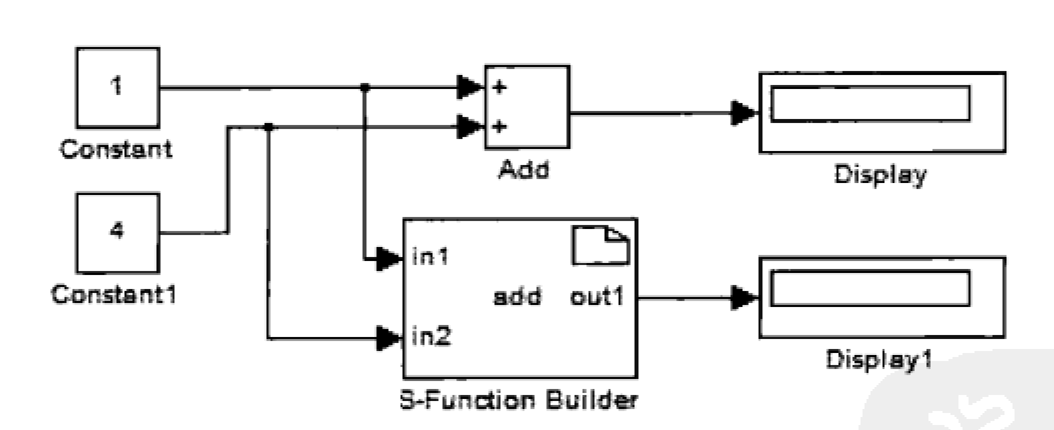


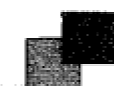
图 4.4.19 S-Function Builder 模型

双击打开 S-Function Builder 模块,作以下设置:

(1) 在 S-function name 字段中写入 add。

(2) 由于本例中不包含连续或离散状态,初始化页面不需要更改。

(3) 在 Data Properties 中,将输入/输出定义为 in1、in2、out1,如图 4.4.20、图 4.4.21 所示。



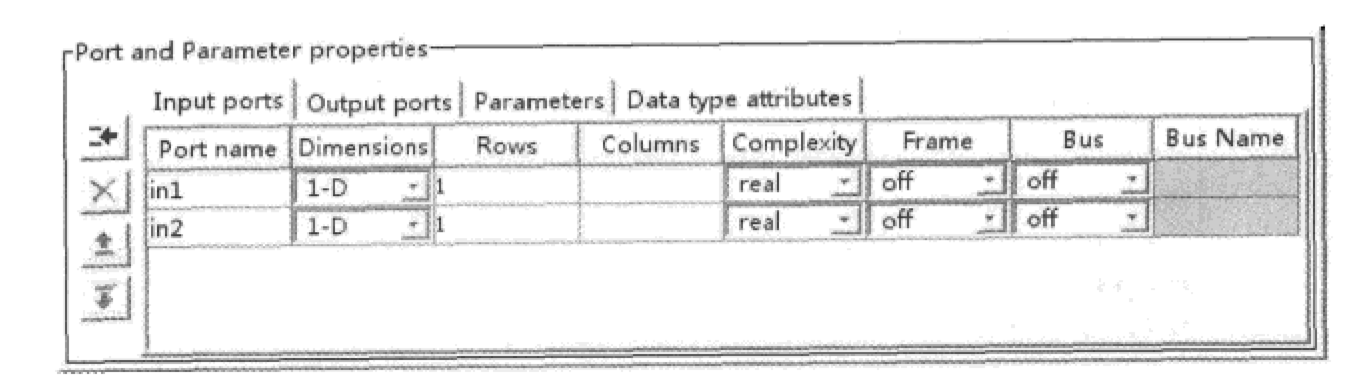


图 4.4.20　Input Ports 界面

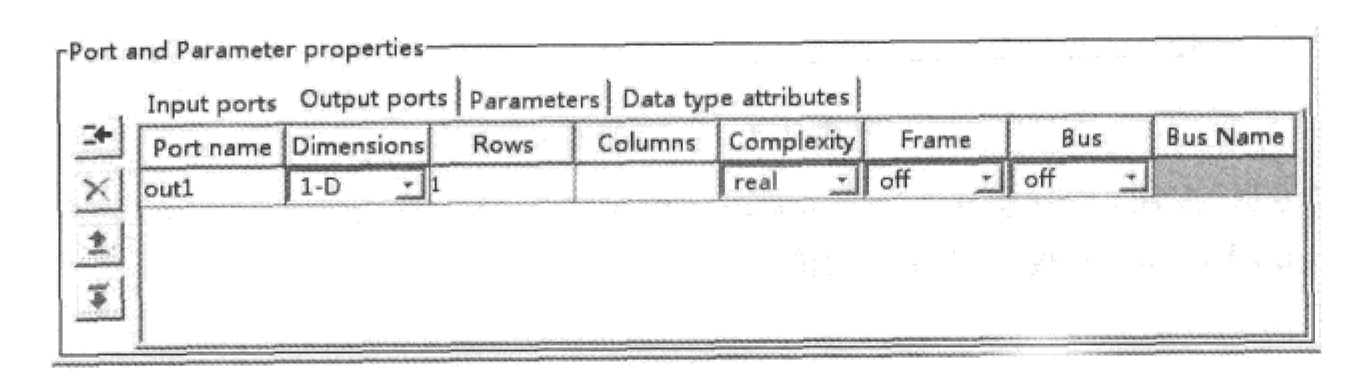


图 4.4.21　Output Ports 界面

(4) 在库文件界面中的 Library/Object/Source files 区域输入 add.c，在 external function declaration 区域输入 extern double add(double in1,double in2)，如图 4.4.22 所示。

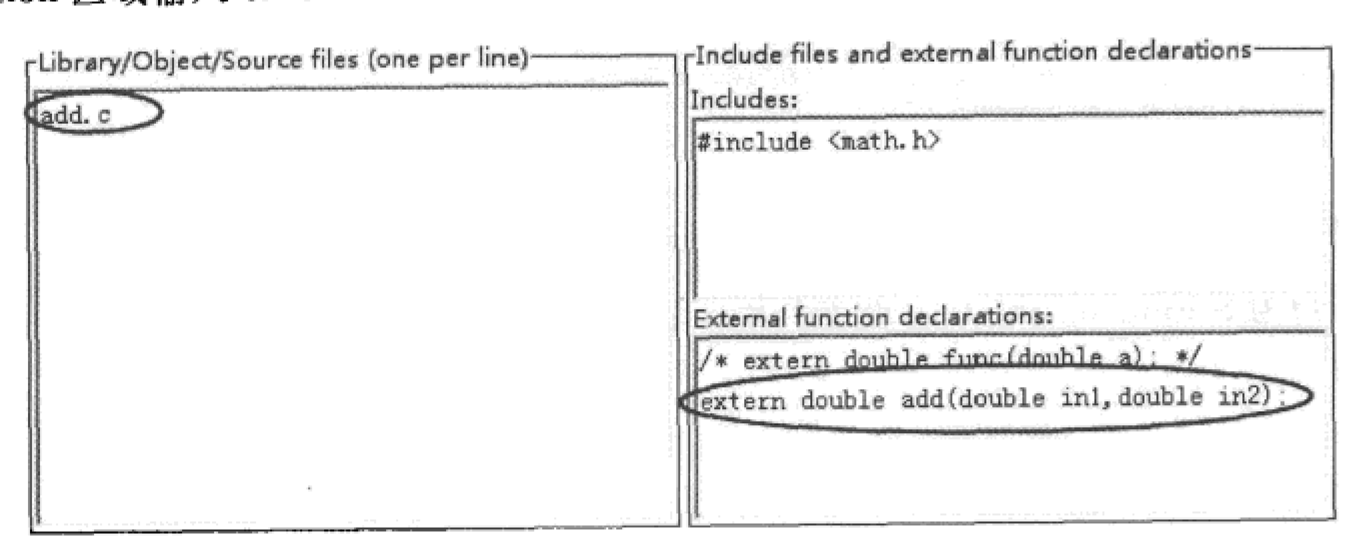


图 4.4.22　库文件界面

(5) 在输出界面中写入 * out1 = add(* in1, * in2)，如图 4.4.23 所示。

Code description

Enter your C-code or call your algorithm. If available, discrete and continuous states should b
xD[0]...xD[n], xC[0]...xC[n] respectively. Input ports, output ports and parameters should be re
symbols specified in the Data Properties. These references appear directly in the generated S

```
*out1=add(*in1,*in2);
```

☑ Inputs are needed in the output function(direct feedthrough)

图 4.4.23　输出界面

(6) 单击对话框工具栏的 Build 按钮,编译信息窗口即显示完成信息,如图 4.4.24 所示。

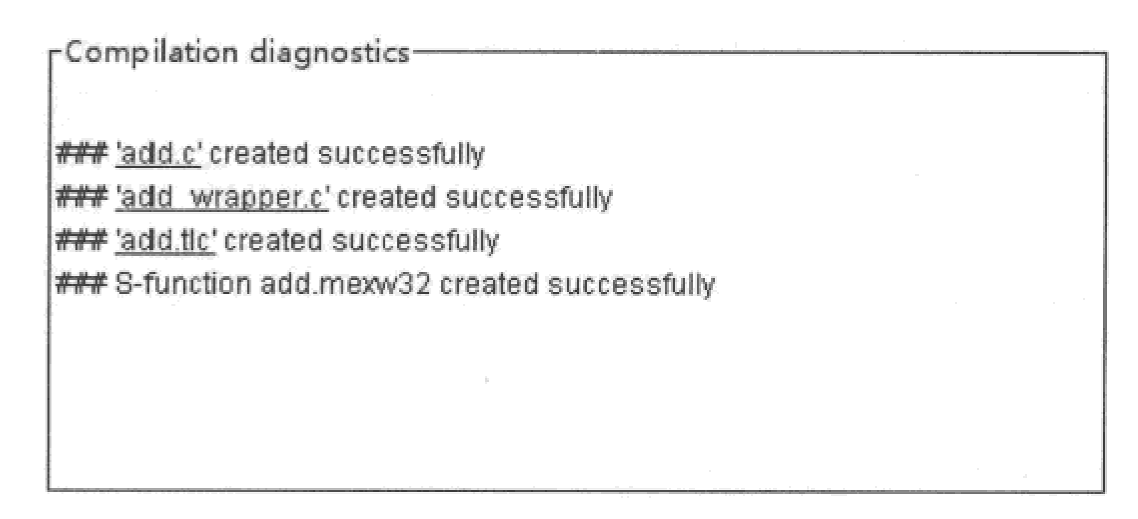


图 4.4.24　编译信息

按下"仿真"按钮,Add 模块与 S-Function Builder 得到一致的结果,如图 4.4.25 所示,说明 S-Function Builder 实现了设计的功能。

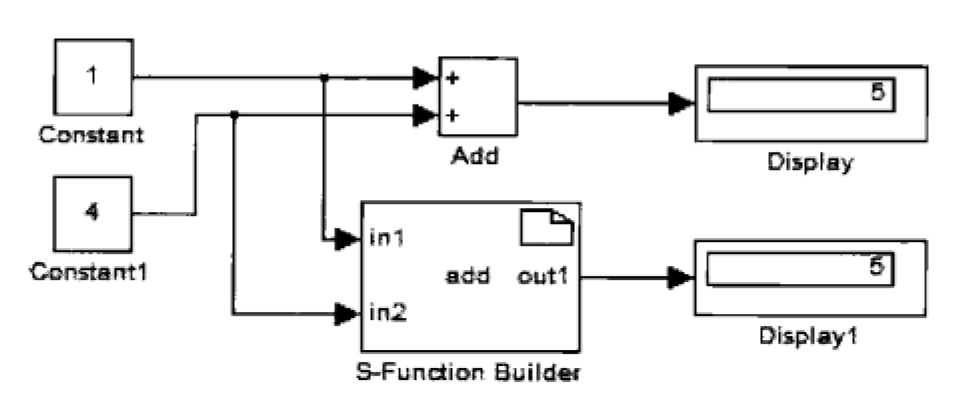


图 4.2.25　仿真结果

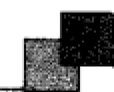
2. S-Function Builder 如何创建 S-Function

当用户为 S-Function Builder 设置参数后,S-Function Builder 将会在当前工作目录下产生下列源文件:sfun. c、sfun_wrapper. c、sfun_tlc. c、sfun_bus. h(其中 sfun 为用户指定的 S 函数名)。之后,S-Function Builder 将会用 mex 指令把上述源代码和用户指定的外部代码、库文件编译成 S 函数。

(1) sfun. c 文件包含有生成 S-Function 标准部分的 C 源代码。

上述例子编译生成的文件为 add. c,部分代码如下:

```
staticvoidmdlInitializeSizes(SimStruct * S)
//mdlInitializeSizes 方法
{……
ssSetNumSFcnParams(S, NPARAMS);//设置参数个数
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
return; //验证参数个数
    }
ssSetNumContStates(S, NUM_CONT_STATES); //设置连续状态个数
ssSetNumDiscStates(S, NUM_DISC_STATES); //设置离散状态个数
……
if (!ssSetNumInputPorts(S, NUM_INPUTS)) return; //验证输入端口个数
```



```
//输入端口 0 属性设置//
ssSetInputPortWidth(S, 0, INPUT_0_WIDTH);
ssSetInputPortDataType(S, 0, SS_DOUBLE);
ssSetInputPortComplexSignal(S, 0, INPUT_0_COMPLEX);
ssSetInputPortDirectFeedThrough(S, 0, INPUT_0_FEEDTHROUGH);
ssSetInputPortRequiredContiguous(S, 0, 1);
//输入端口 1 属性设置//
ssSetInputPortWidth(S, 1, INPUT_1_WIDTH);
ssSetInputPortDataType(S, 1, SS_DOUBLE);
ssSetInputPortComplexSignal(S, 1, INPUT_1_COMPLEX);
ssSetInputPortDirectFeedThrough(S, 1, INPUT_1_FEEDTHROUGH);
ssSetInputPortRequiredContiguous(S, 1, 1);

if (!ssSetNumOutputPorts(S, NUM_OUTPUTS)) return; //验证输出端口个数
//输出端口 0 属性设置//
ssSetOutputPortWidth(S, 0, OUTPUT_0_WIDTH);
ssSetOutputPortDataType(S, 0, SS_DOUBLE);
ssSetOutputPortComplexSignal(S, 0, OUTPUT_0_COMPLEX);
ssSetNumSampleTimes(S, 1);
ssSetNumRWork(S, 0);
ssSetNumIWork(S, 0);
ssSetNumPWork(S, 0);
ssSetNumModes(S, 0);
ssSetNumNonsampledZCs(S, 0);
…
staticvoidmdlInitializeSampleTimes(SimStruct * S) //mdlInitializeSampleTimes 方法
{
ssSetSampleTime(S, 0, SAMPLE_TIME_0);  //设置采样时间
ssSetOffsetTime(S, 0, 0.0);            //设置偏置时间
}
…
staticvoidmdlOutputs(SimStruct * S, int_Ttid)
//mdlOutputs 方法
{
constreal_T * in1 = (constreal_T * ) ssGetInputPortSignal(S,0);
constreal_T * in2 = (constreal_T * ) ssGetInputPortSignal(S,1);
real_T       * out1 = (real_T * )ssGetOutputPortRealSignal(S,0);
add_Outputs_wrapper(in1, in2, out1);
}
staticvoidmdlTerminate(SimStruct * S)
//mdlTerminate 方法
{
}
```

(2) sfun_wrapper. c 文件包含有在 S-Function Builder 对话框中输入的用户代码。

编译生成文件为“add_wrapper. c”，部分代码如下：

```
……
externdouble add(double in1,double in2); //声明外部函数“add”
voidadd_Outputs_wrapper(constreal_T * in1,
constreal_T * in2,
real_T * out1)
//调用用户代码
{
 * out1 = add( * in1, * in2);
}
```

(3) sfun. tlc 文件能够使生成的 S-Function 在 Simulink 中以 Rapid Accelerator 模式运行，并且在生成代码时允许内联 S-Function。此外，本文件还能生成 Accelerator 模式下的 S-Function，用以加速模型运行。

编译生成文件为“add. tlc”，部分代码如下：

```
 % implements add "C"
 % function BlockTypeSetup(block, system) Output
//BlockTypeSetup 函数
   % openfile externs

extern void add_Outputs_wrapper(constreal_T * in1,
constreal_T * in2,
real_T * out1);
   % closefile externs
   % < LibCacheExtern(externs)>
   % endfunction

 % function Outputs(block, system) Output
//Outputs 函数
     % assign pu0  =  LibBlockInputSignalAddr(0, "", "", 0)
   % assign pu1  =  LibBlockInputSignalAddr(1, "", "", 0)
   % assign py0  =  LibBlockOutputSignalAddr(0, "", "", 0)
   % assign py_width  =  LibBlockOutputSignalWidth(0)
   % assign pu_width  =  LibBlockInputSignalWidth(0)
add_Outputs_wrapper( % < pu0 >,  % < pu1 >,  % < py0 >);
   % endfunction
```

(4) sfun_bus. h。当用户在 S-Function Builder 的 Data Properties 页面中指定输入/输出为总线时，S-Function Builder 会在缺省头文件时自动生成该头文件。

由于在上节的例子中并未将输入/输出指定为总线，因此并未产生 sfun_bus. h 文件。

第 5 章

8051 单片机代码的快速生成

本章重点讲述基于模型设计的 8051 单片机快速开发。自动代码生成工具 Real-time Workshop Embedded Coder 的设置，在作者所著的《基于模型的设计及其嵌入式实现》一书中已做了详细介绍，有需要的读者可以阅读相关内容或参考 MathWorks 公司的技术手册。

一个完整的基于模型设计的例子需要数 10 页的篇幅来进行介绍，为了给读者呈现更多的开发实例，这里仅做模型的功能验证、简单的模型数据类型变换、软件在环测试、处理器在环测试、嵌入式实时 C 代码的快速生成。考虑到多数读者不可能拥有各种类型的开发板，本章的硬件/硬件在环测试将采用 Proteus 虚拟硬件测试平台进行，完整的基于模型设计的例子将在第 9 章介绍。

本章推荐读者采用 MathWorks R2010b 和 TASKING EDE for 8051 v7.2 r1 进行基于模型的开发。这样既能提高生成代码的效率，还可以减少后期修改代码的工作量。如果读者不习惯 TASTKING EDE 软件，也可以采用国内流行的 Keil C51 进行基于模型的设计。

本章的主要内容：

- Proteus 快速入门。
- Keil C51 集成开发环境。
- TASKING 嵌入式开发环境。

5.1 仿真软件 Proteus 快速入门

5.1.1 Proteus 简介

Proteus 是英国 Labcenter 公司开发的电路分析与实物仿真软件。它集成了 ISIS 原理图绘制和 ARES PCB 制版，是一款功能强大的 EDA 软件，同时具备完善的仿真功能，能够支持目前流行的大部分单片机系统，如 51 系列、MSP430、AVR、PIC、ARM7 等常用主流芯片。还可以直接在基于原理图的虚拟原型上编程，再配合系统配置的虚拟逻辑分析仪、示波器等，看到系统输出效果。Proteus 建立了一套完整的虚拟产品开发平台，并且可以和第三方编译软件联调，如 Keil、IAR、MPLAB 等。

Proteus 基本功能：

(1) 智能原理图输入系统(ISIS)。

① 器件库。超过 27 000 种元器件，且用户可自定义器件。

② 器件搜索。通过模糊搜索功能，可以快速找到用户所需要的器件。

③ 自动连线。智能的连线功能使连线简单快捷，降低用户的劳动强度。

④ 总线结构。采用总线器件和总线布线使电路一目了然。

⑤ 多种格式的输出。个性化的设置，可输出高质量的 BMP 图纸，也可生成 Word、Powerpoint 等格式的文档。

(2) 混合模型 SPICE 仿真(ProSPICE)。

① ProSPICE。采用工业标准 SPICE3F5，实现数-模电路的混合仿真。

② 仿真元件库。Proteus 目前拥有超过 27 000 种仿真器件，Labcenter 公司仍在不断努力提供更多新的仿真器件。同时，用户也可通过内部原型或厂家的 SPICE 文件自定义仿真元件，还可导入第三方的仿真元件。

③ 激励源。包括直流、正弦、脉冲、分段线性脉冲、音频、指数信号、单频 FM、数字时钟和码流，以及文件形式的信号输入。

④ 虚拟仪器。包括示波器、逻辑分析仪、信号发生器、直流电压/电流表、交流电压/电流表、频率计/计数器、逻辑探头、虚拟终端、SPI 调试器、I^2C 调试器等，其优点是用户可以随心所欲地使用这些虚拟仪器，不再受实际仪器的功能限制和价格限制。

⑤ 仿真显示。用色点标示引脚电平的高低，以不同颜色表示导线上电压的大小，采用喇叭、蜂鸣器、电动机、LED、液晶显示屏等，使仿真结果更加逼真与生动。

⑥ 高级图形仿真功能。采用图标，精确分析工作点、瞬态特性、频率特性、传输特性、噪声、失真、频谱等多项电路性能指标。

(3) 虚拟系统模型(VSM)。

① 支持 CPU 的种类。8051/52、MSP430、AVR、PIC10/12、PIC16、PIC18、PIC24、dsPIC33、HC11、BasicStamp、ARM7、8086 等，Proteus 8.0 版还将支持 ARM Cortex、DSP 等处理器。

② 支持的外设。包括键盘/按键、电动机、LCD 模块、LED 点阵、LED 七段显示模块、RS232 虚拟终端、电子温度计等。且 COMPIM 还可通过 PC 串口和仿真电路实现双向异步串行通信。

③ 实时仿真：支持中断、SPI/I^2C、MSSP、PSP、RTC、UART/USART/EUSARTs、ADC、CCP/ECCP 等仿真。

④ 支持联调。支持汇编语言的编辑/编译/源码级仿真，内带 8051、AVR、PIC 的汇编编译器，也可以与第三方集成编译环境(如 IAR、Keil、MPLAB)无缝连接，实现联调功能，就像接上仿真器一样方便。

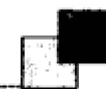
(4) PCB 制作。

① 原理图到印制电路板的一键功能。原理图绘制完成，再经过仿真验证其功能后，便可一键转入 ARES 的 PCB 设计环境，大大简化后续的设计。

② 自动布局/布线。支持器件的手工/自动布局、无网格自动布线或手工布线、引脚交换/门交换等功能。

③ 优异的 PCB 设计功能。支持多达 16 个铜箔层，2 个丝印层，4 个机械层，支持 3D 动画以找出设计中难以发现的瑕疵，灵活的布线方案供用户选择，支持自动设计规则检查等。

④ 多种输出格式。支持 Gerber 文件的导入或导出，支持与其他 PCB 设计软件的转换、设计和加工。



下面将以 Proteus 7.7 sp2 版为例讲述 Proteus 的使用方法。

5.1.2　快速绘制原理图

绘制原理图是 Protues 仿真软件的基础，本节将在 ISIS 原理图绘制系统中完成一个实际的单片机仿真电路。

(1) 双击 ISIS 图标，打开原理图绘制界面，出现图 5.1.1 所示的默认模板，下面的例子将在默认模板中完成。

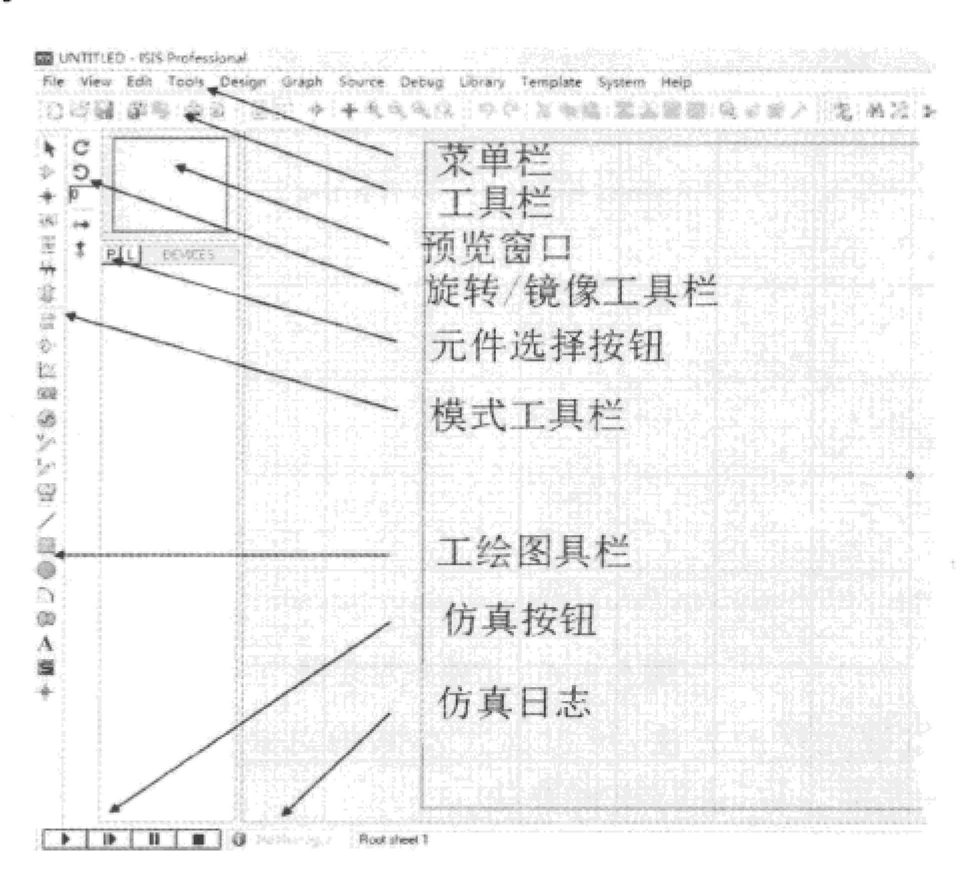


图 5.1.1　原理图绘制界面

比较熟悉 Proteus 的用户也可以选择其他更合适自己的模板。在菜单中选择 File→New Design 命令，弹出的对话框中列出了所有可用模板，如图 5.1.2 所示。

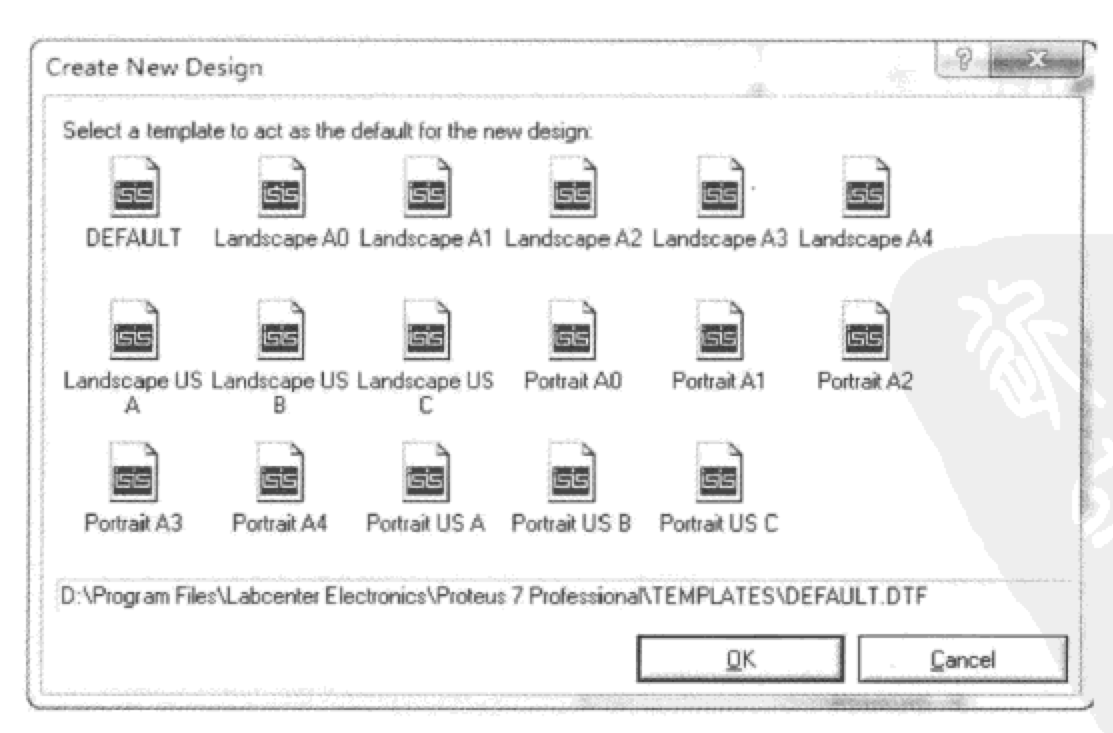


图 5.1.2　选择模板

(2) 单击按钮或选择菜单项 Library→Pick Device/Symbol，在 Pick Device 对话框中选择所需的元件，如图 5.1.3 所示。

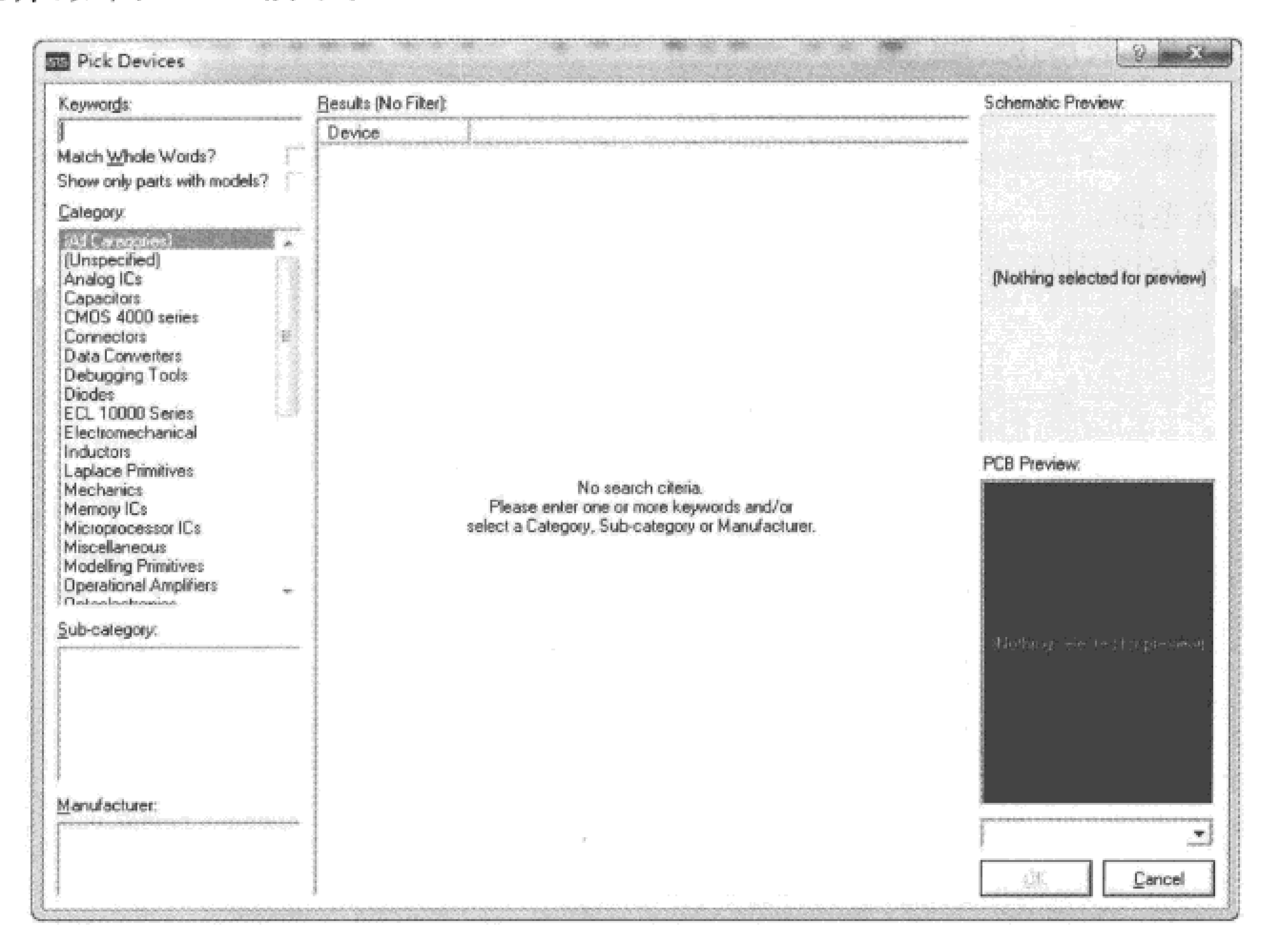


图 5.1.3　元件库对话框

ISIS 提供了两种选取元件的方法：可以通过左侧的目录区逐级向下查找，也可在左上方 Keywords 区域中输入元件名称，进行模糊查找。

例如，要选择 atmel 公司的 89C51 芯片，可在 Keywords 中输入 8951，则在 Result 列表中会显示所有名称中含有 8951 的元件，用户选择所需的一款元件即可。如图 5.1.4 所示。

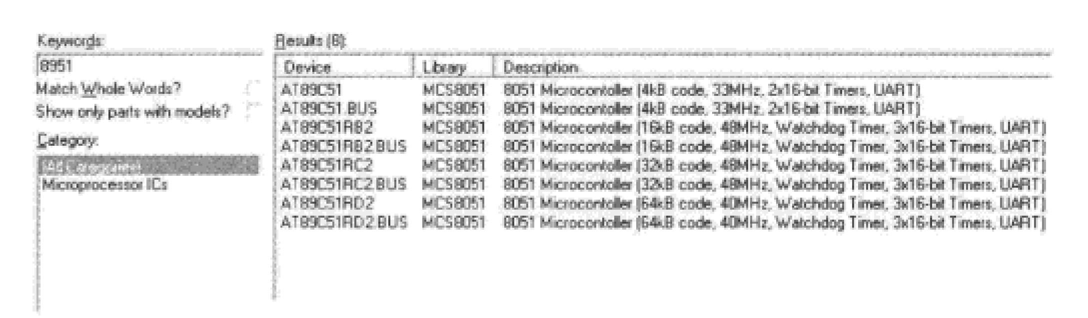


图 5.1.4　元件筛选

选择好元件后单击 OK 按钮，所选元件即出现在左侧的 Devices 列表中，并且在预览窗口中显示该元件的预览图。如图 5.1.5 所示。

其他元件可用同样的方法添加到 Devices 列表中。本例需要添加的元件，如图5.1.6所示。

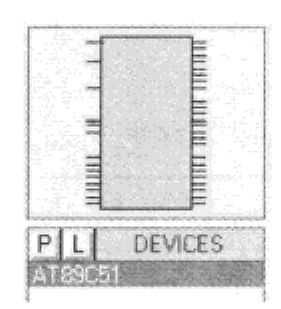


图 5.1.5　元件预览

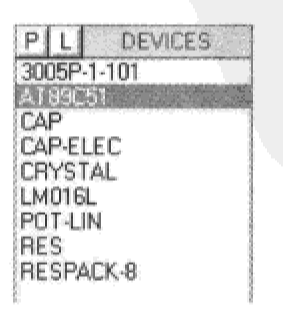
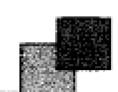


图 5.1.6　本例所需元件列表



(3) 元件选择完毕后，下一步的工作是摆放元件。在 Device 列表中选择一款元件，例如 89C51，并在绘图区单击，出现一个红色框图，用户可将其调整到合适位置再次单击，摆放该元件，如图 5.1.7 所示。

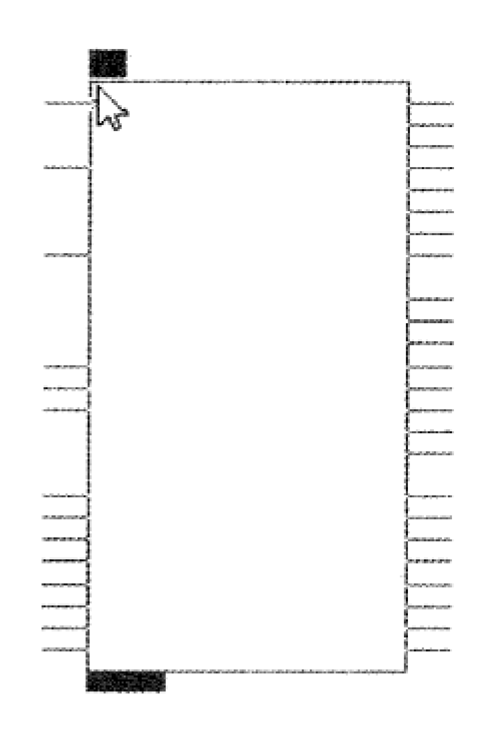


图 5.1.7 添加元件

用户可按同样方法将其他元件摆放在合适的位置。需要注意的是，某些元件的默认摆放方向并不合适，可以使用窗口左上方的旋转工具栏按钮或进行调整。例如晶振 CRYSTAL 初始为水平方向，通过旋转工具栏按钮将其调整为垂直方向，如图 5.1.8 所示。

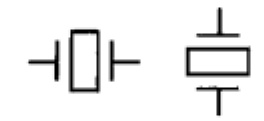


图 5.1.8 元件方向调整

元件摆放如图 5.1.9 所示。

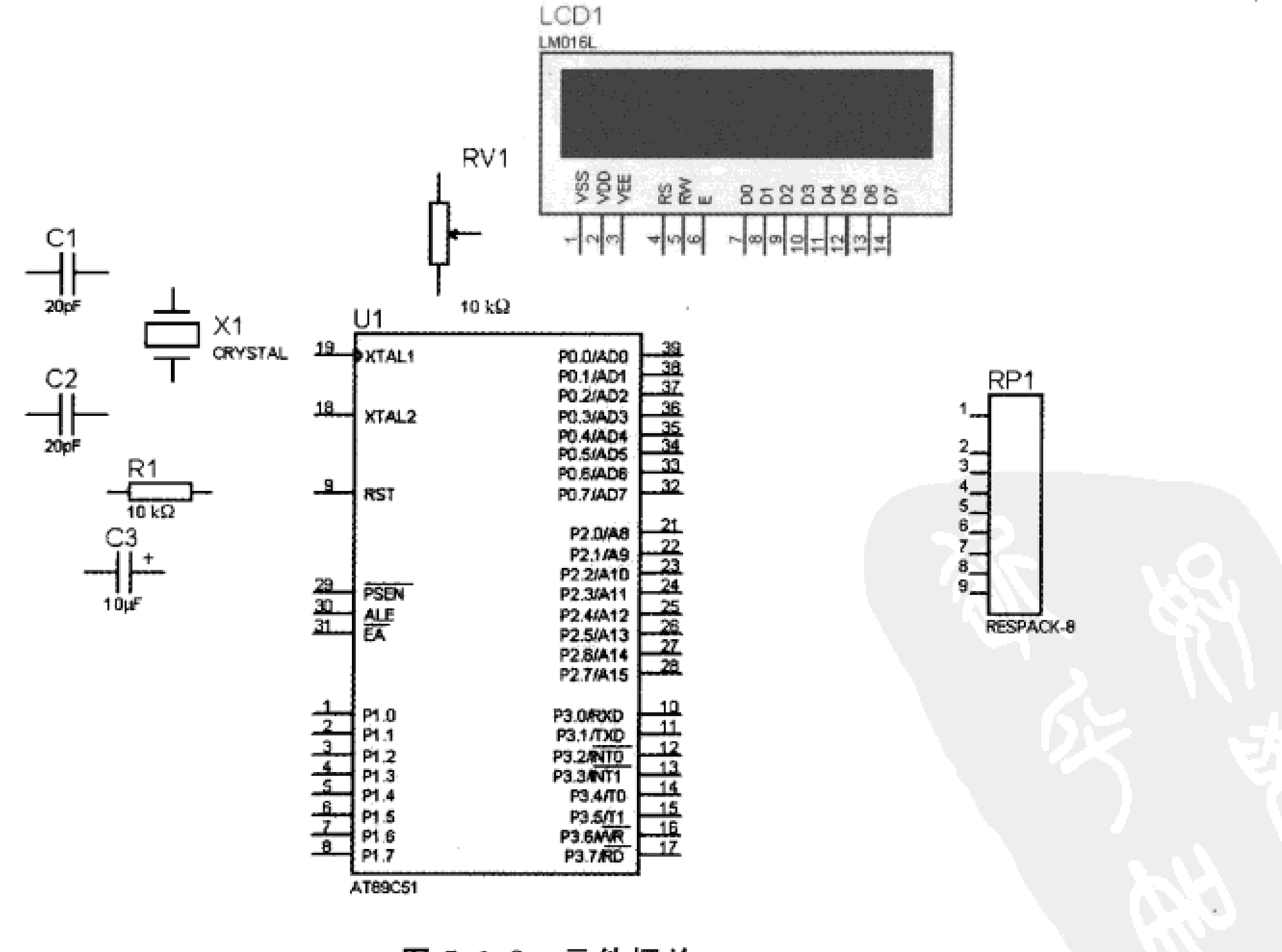
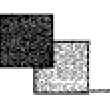


图 5.1.9 元件摆放



除了所选元件外，在原理图中一般还需用到电源与接地等终端，可通过模式工具栏的终端按钮添加。单击按钮后，在 Device 列表中列出了所有可用终端。如图 5.1.10 所示。

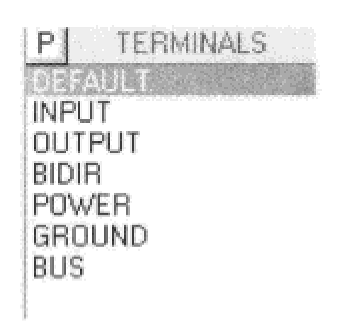


图 5.1.10 终端模块

将电源 POWER、接地 GROUND 添加到绘图区域中合适的位置，如图 5.1.11 所示。

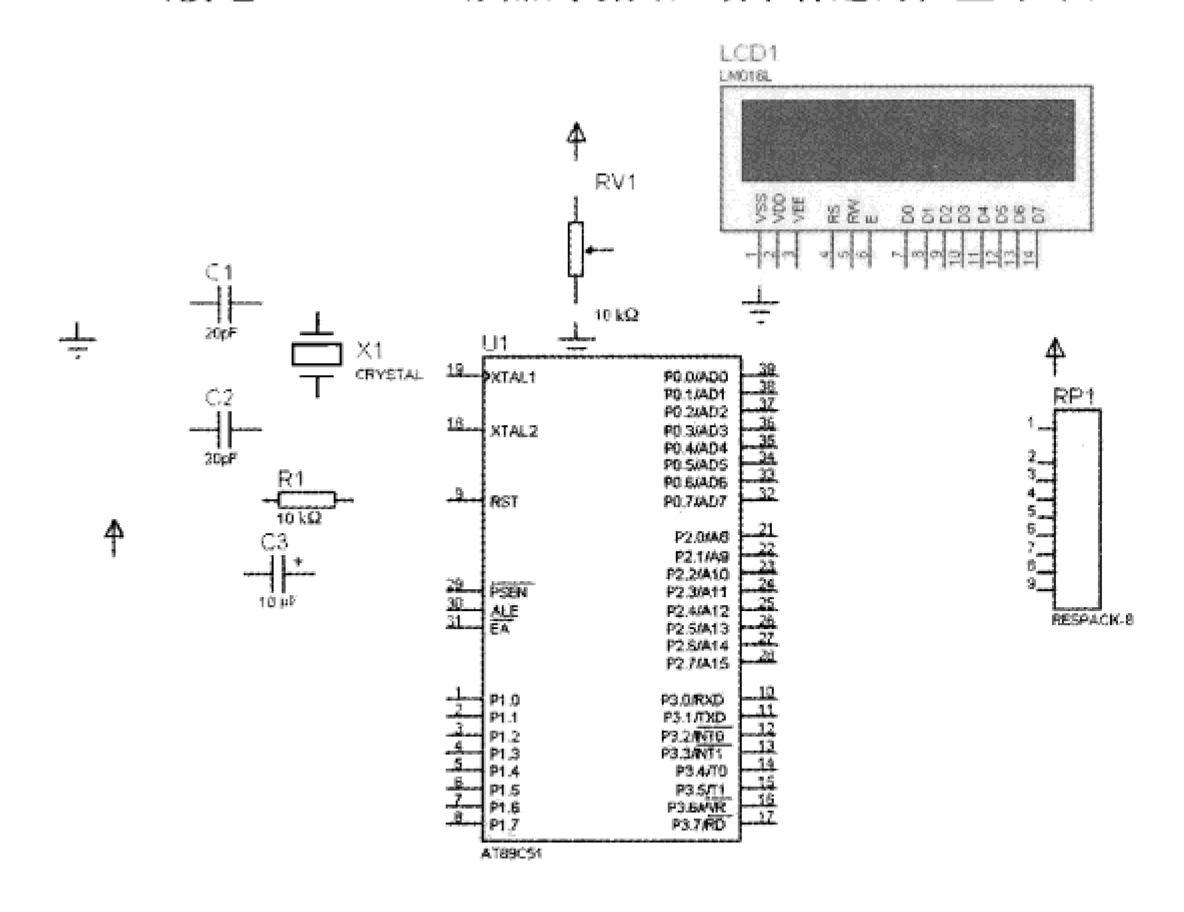


图 5.1.11 添加电源与接地

(4) 元件添加到绘图区后，双击元件，可打开元件编辑窗口，如图 5.1.12 显示的是电容 C1 的编辑窗口。

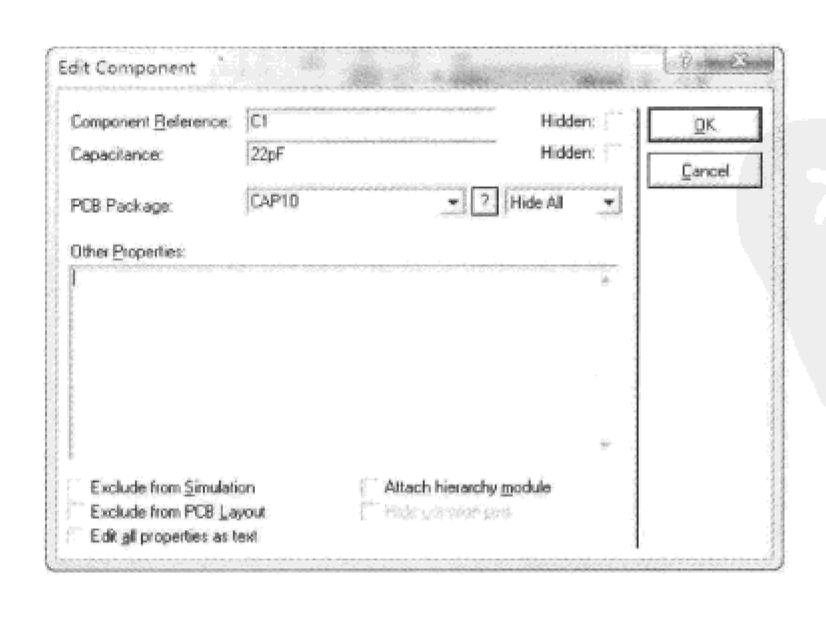


图 5.1.12 元件属性编辑窗口

（5）属性设置：

Conponent Reference：该元件在原理图中的编号为C1。

Capacitance：该电容的电容值为22pF。

PCB Package：该元件的封装格式为CAP10。

如果选中Hidden，则对应的属性就不会在原理图中显示。配置完成后单击OK按钮确认。用同样的方法配置C2为22pF，C3为10pF，R1为10 kΩ，RP1为220 Ω，RV1为10 kΩ。

（6）元件摆放完毕后，需要进行的是连线工作。在Proteus中可以很方便地完成走线，它同时支持自动与手动布线，用户只需选择起点和终点，系统会自动走出一条合适的连线，再结合手动布线可以达到很好的布线效果。

例如芯片左侧的晶振电路部分，在GROUND的引脚处单击，并向电容C1移动，系统会自动走出图5.1.13所示的连线，然后在C1的左侧引脚处再次单击，完成连线工作。

如果用户想要自定义走线的路径，可以在期望的拐点处单击，如图5.1.14所示。

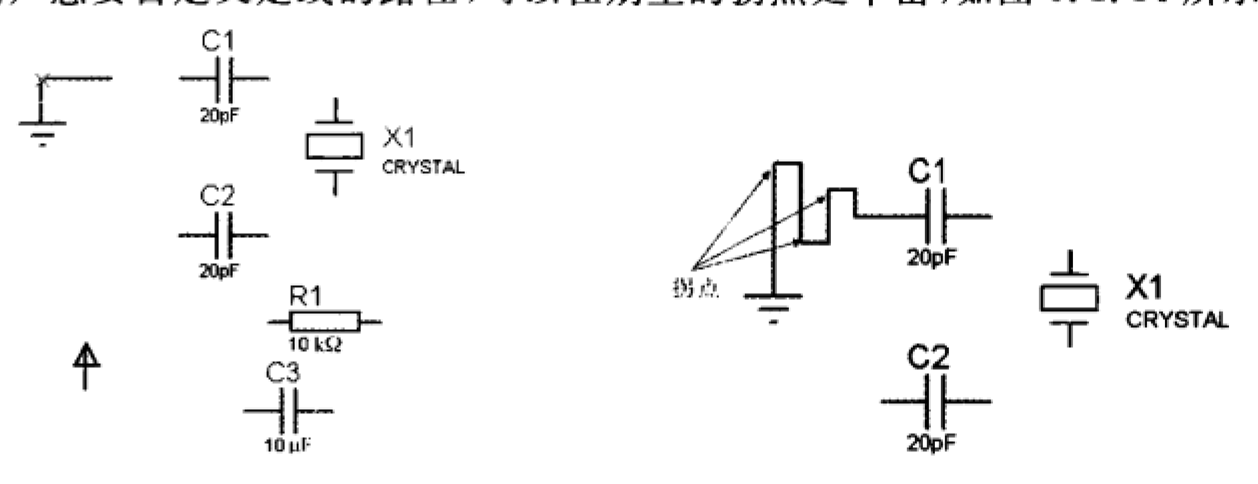


图5.1.13　自动连线　　　图5.1.14　自定义连线

在元器件引脚数目较大时，使用总线可以有效地简化原理图。用户可通过按钮添加总线，总线的放置步骤为：在期望的总线起始点、期望的拐点处单击，最后在期望的截止点双击。如图5.1.15所示。

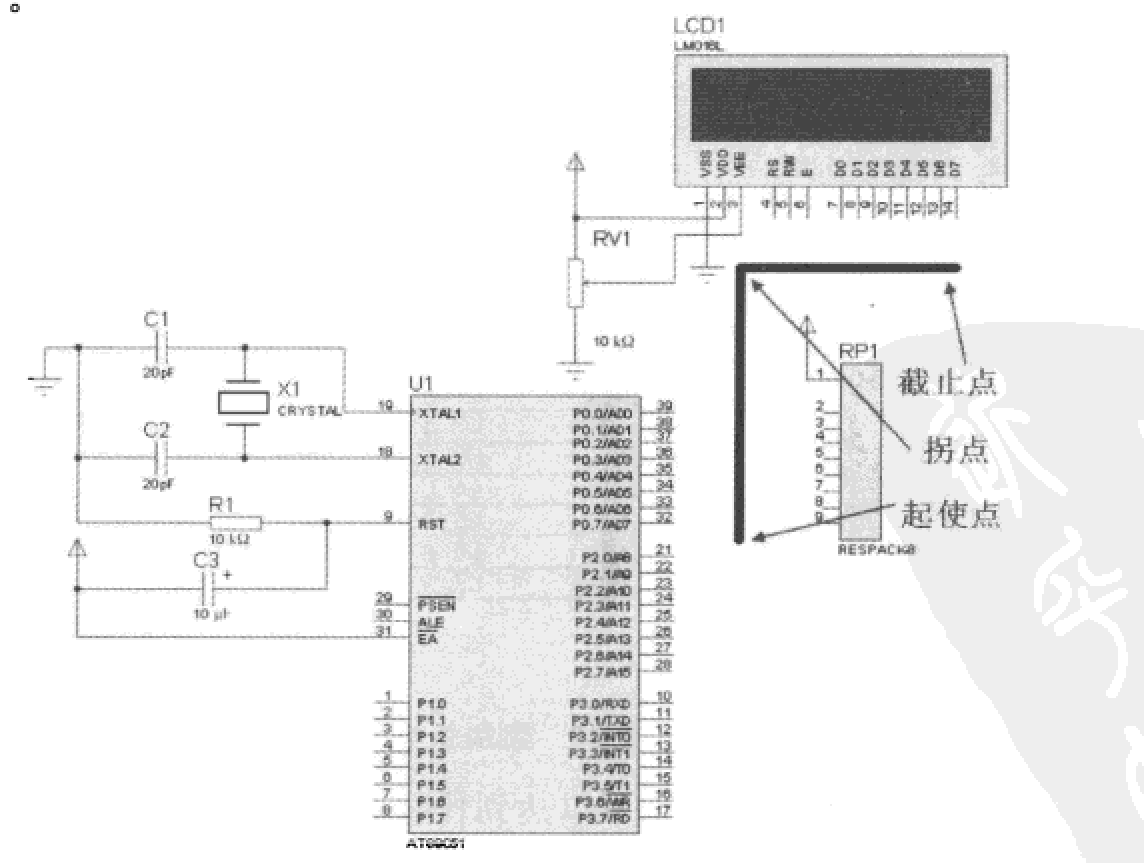


图5.1.15　添加总线

放置总线后，可将引脚连接到总线上。需要注意的是，使用总线后，需要为连接到总线上的连线添加标签以使其一一对应。可先选中要添加标签的连线，右击，在右键菜单中选择 Place Wire Label 即可添加标签，如图 5.1.16 所示。

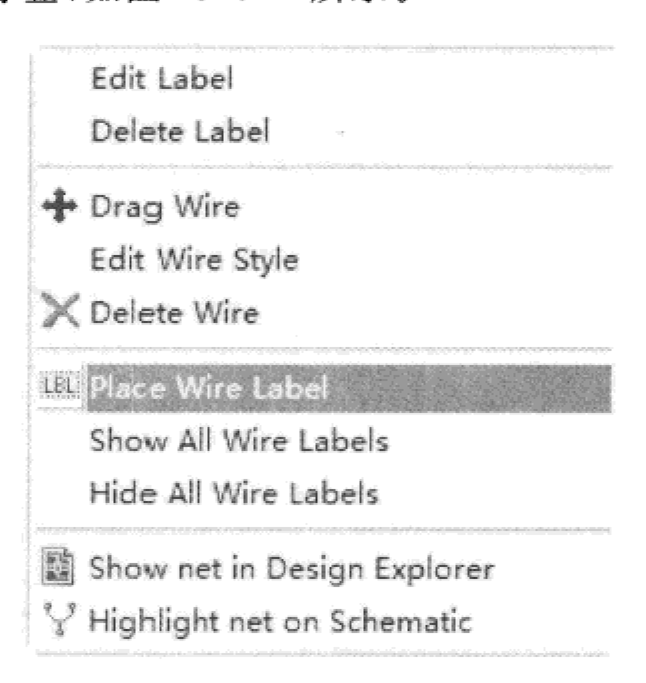


图 5.1.16　添加标签菜单

本例中，为总线添加标签 L[0...7]，表示其集成了 L0～L7 这 8 条连线。然后为连接到总线上的连线分别添加标签 L0～L7，如图 5.1.17 所示。

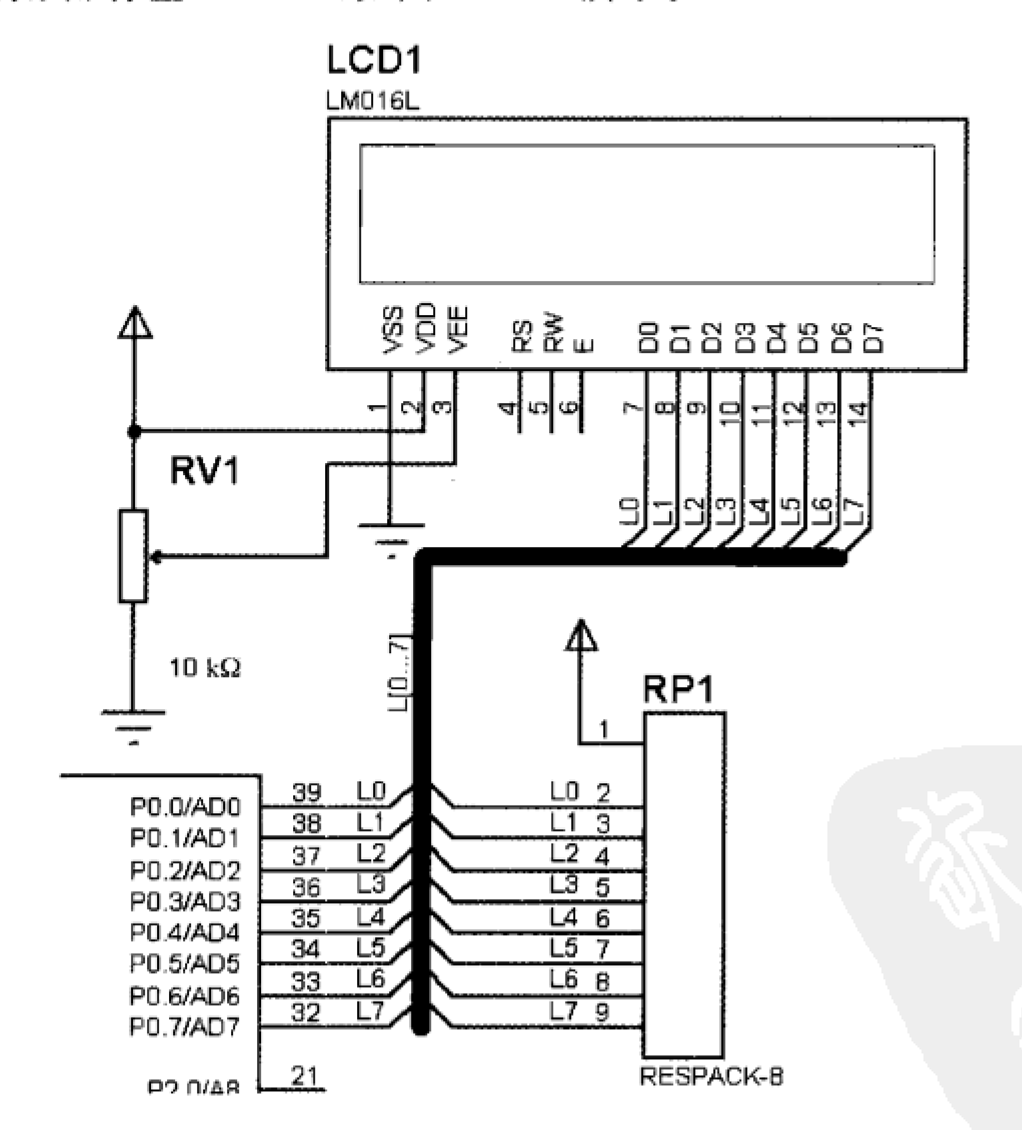
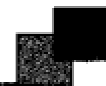


图 5.1.17　添加总线标签

除总线外，为单个引脚添加标签同样可以达到代替直接连线、简化原理图的效果。例如，可通过标签将 LCD 的 4、5、6 号引脚与 89C51 的 P2.0、P2.1、P2.2 引脚相连，如图 5.1.18 所示。



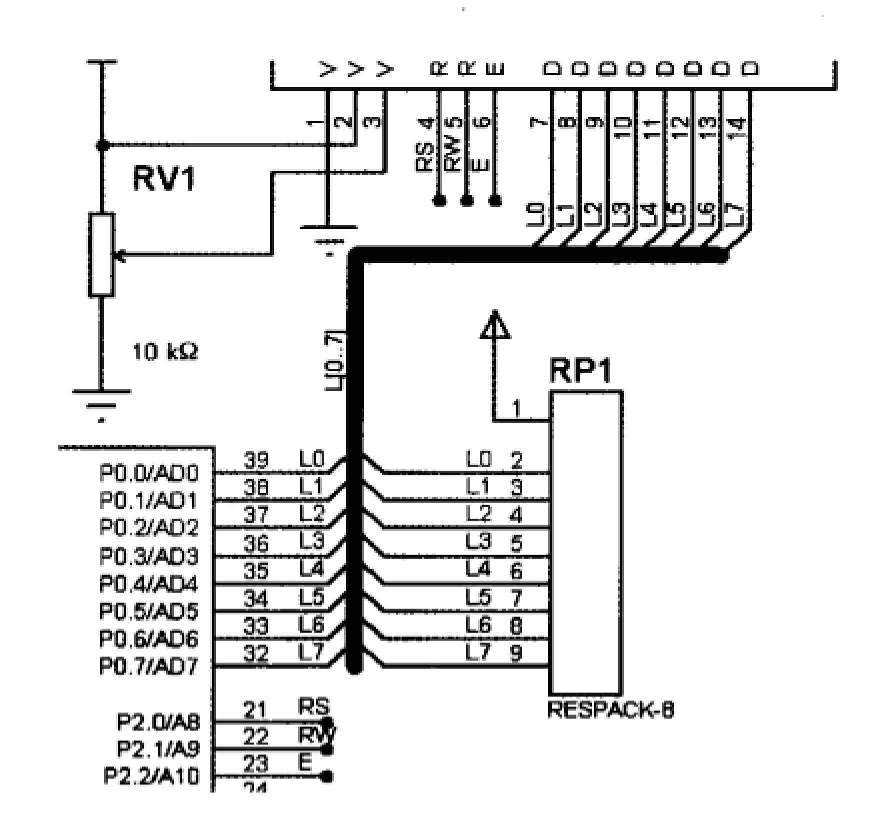


图 5.1.18　添加单一连线标签

布线完毕后的原理图如图 5.1.19 所示。

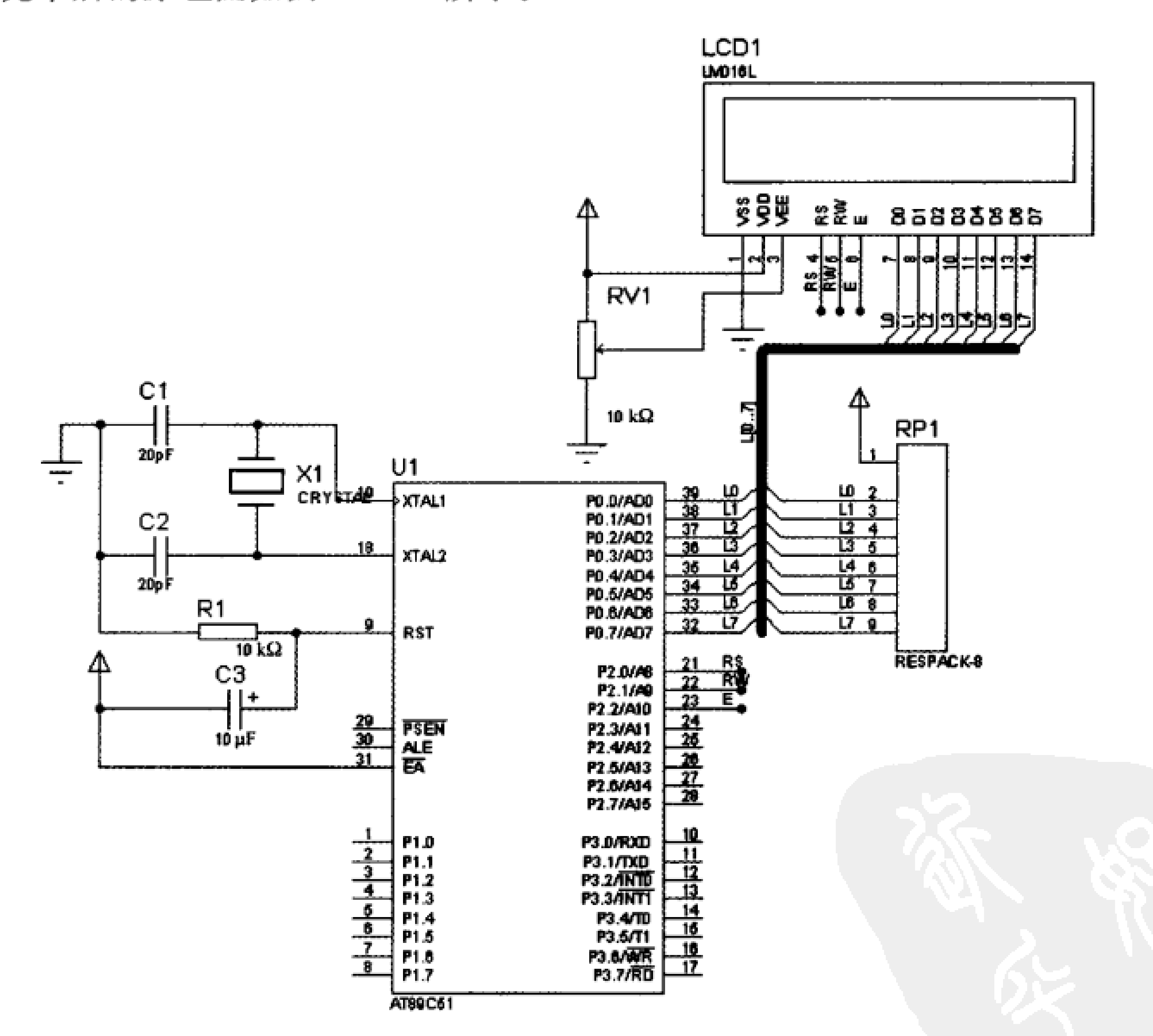


图 5.1.19　布线完毕后的原理图

5.1.3 PCB 制板

完成原理图绘制及相关属性设置后，单击按钮▩，可一键切换到 ARESPCB 制板软件，对原理图进行制板前的处理工作。ARES 支持自动/手动布局、自动/手动布线，结合其内部集成的大量元器件封装，可以方便地由原理图绘制出 PCB 设计图，并且能够输出 PCB 板的 3D 图像，使用户更直观地看到可能存在的不合理布局，如图 5.1.20 所示。

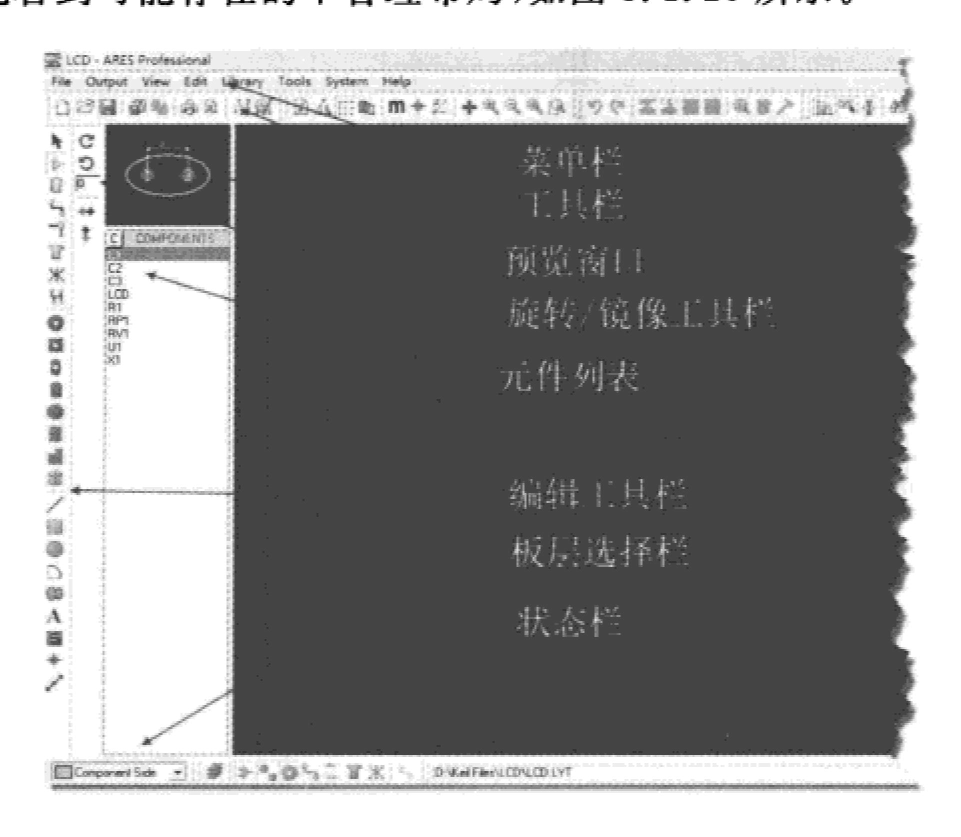


图 5.1.20 PCB 图绘制界面

(1) 在布板前，首先要设置板的边框尺寸，选择编辑工具栏中的 2D 矩形按钮▩，并在板层选择栏中选择 Board Edge。在绘制板的边框前，还需要注意工具栏上的**m**按钮，转换英制/米制单位，这里选择米制单位。

完成上述工作后，按 O 键，定位坐标原点。按住鼠标左键并拖动，下方的状态栏会显示鼠标指针当前所在的坐标，在合适的位置再次单击，即完成边框的绘制工作。本例中绘制一个 80×55 mm 的矩形框，如图 5.1.21 所示。

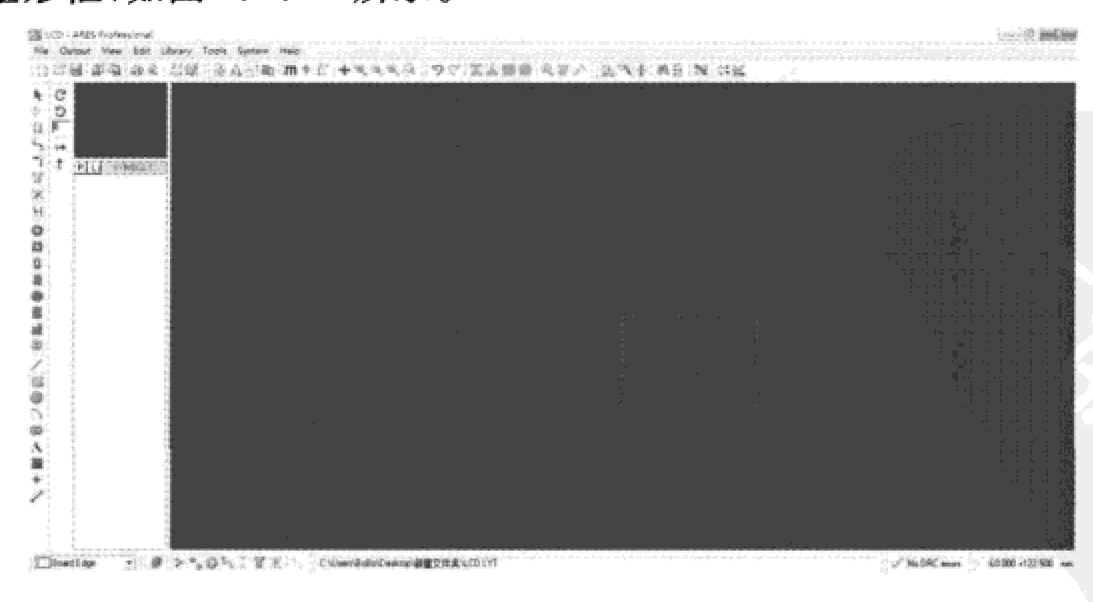
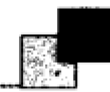


图 5.1.21 绘制边框



(2) 单击编辑工具栏按钮 ，元件列表中会列出原理图中所用到的所有元件，并在预览窗口中显示元件封装格式，如图 5.1.22 所示。

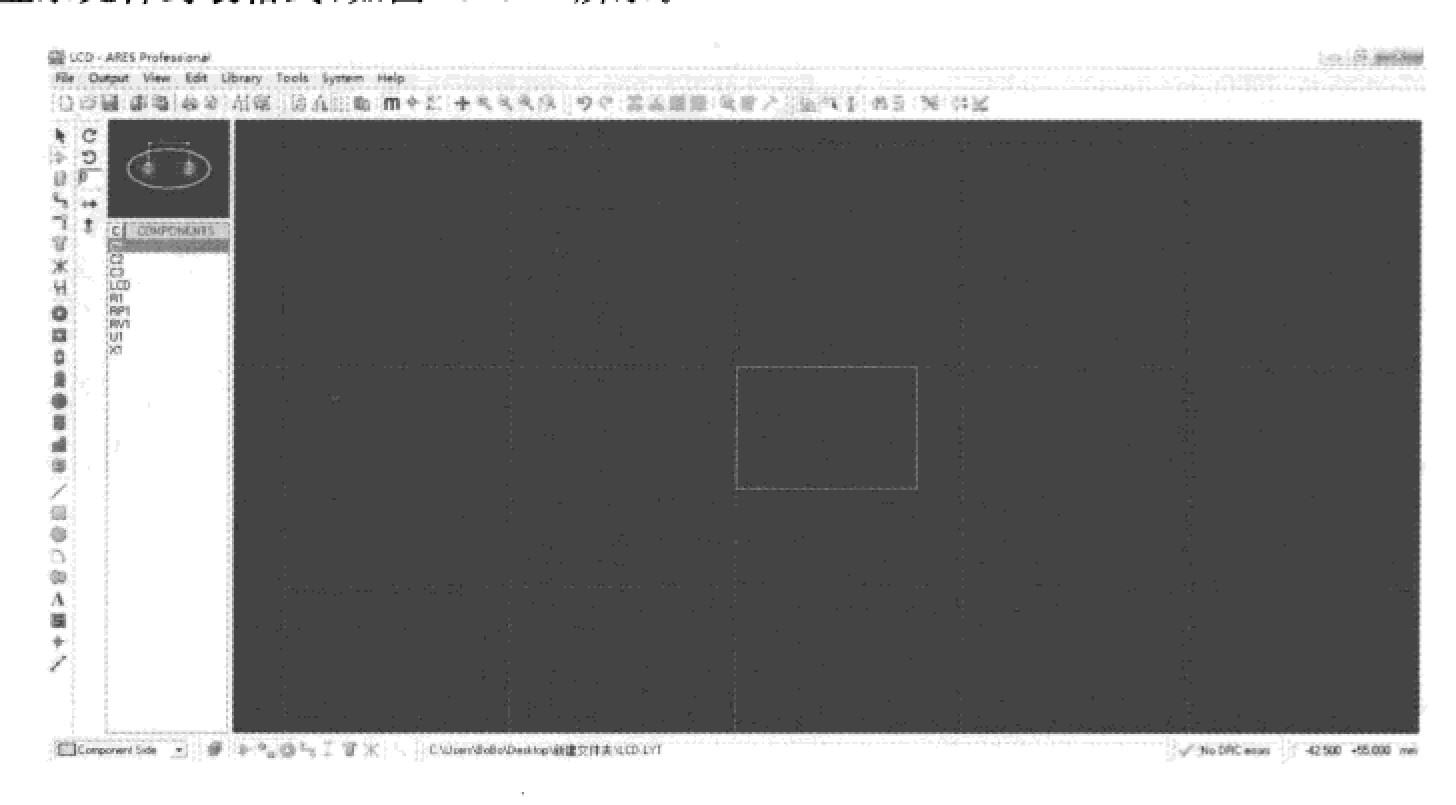


图 5.1.22　预览元件封装

本例中所用元件较少，因此采用手工布局。首先单击工具栏中的按钮 ，将绘图区调整到合适的大小，然后依次将列表中的元件摆放到边框之内，如图 5.1.23 所示。

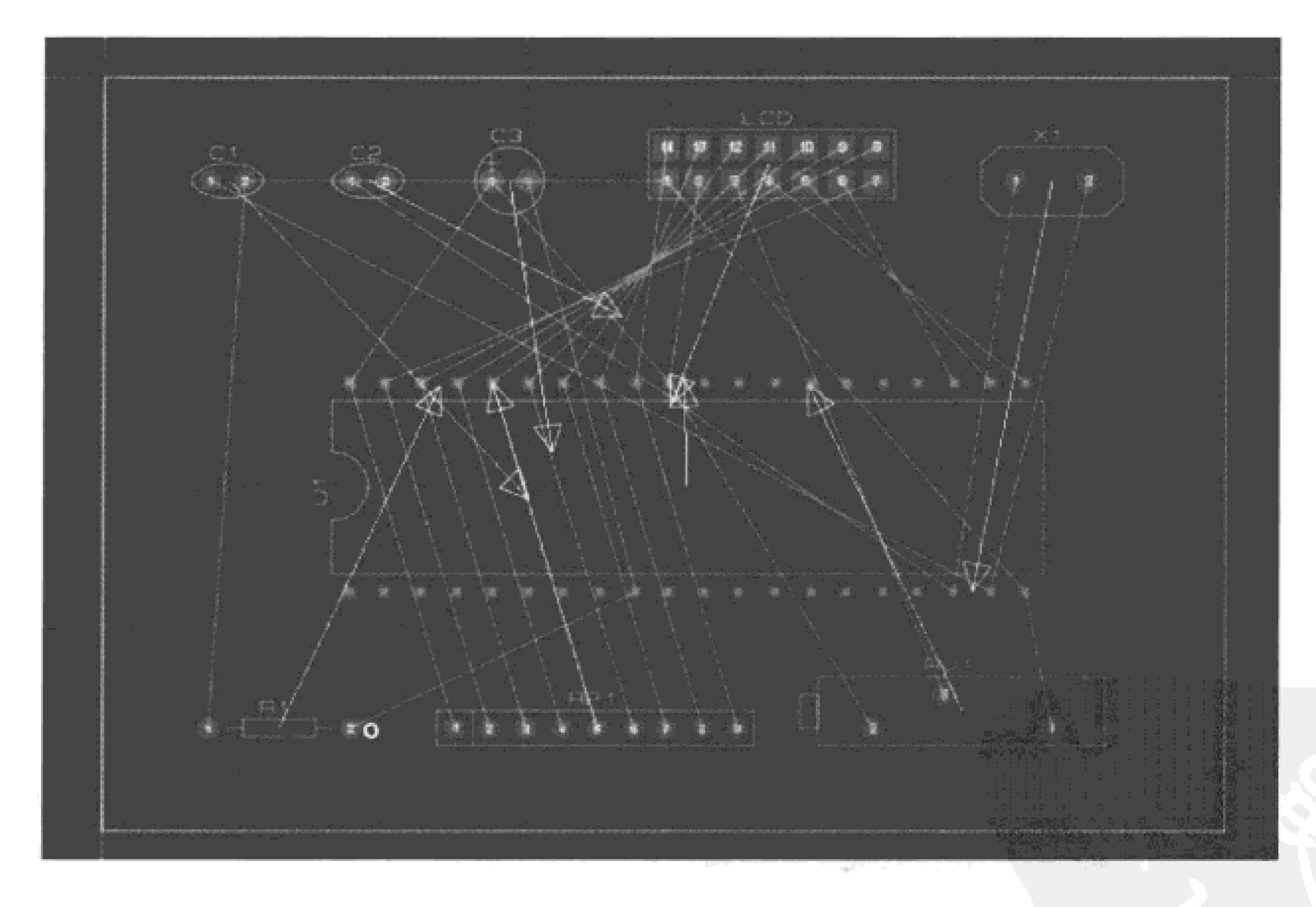


图 5.1.23　手工摆放元件

(3) 元件摆放完毕后，利用 ARES 的自动布线功能可以快速完成布线工作，必要的时候结合手动布线，可以得到一个令人满意的结果。

单击工具栏按钮 ，弹出图 5.1.24 所示的自动布线对话框，本例选用默认设置，单击 Begin Routing 按钮，开始布线(图 5.1.25)。

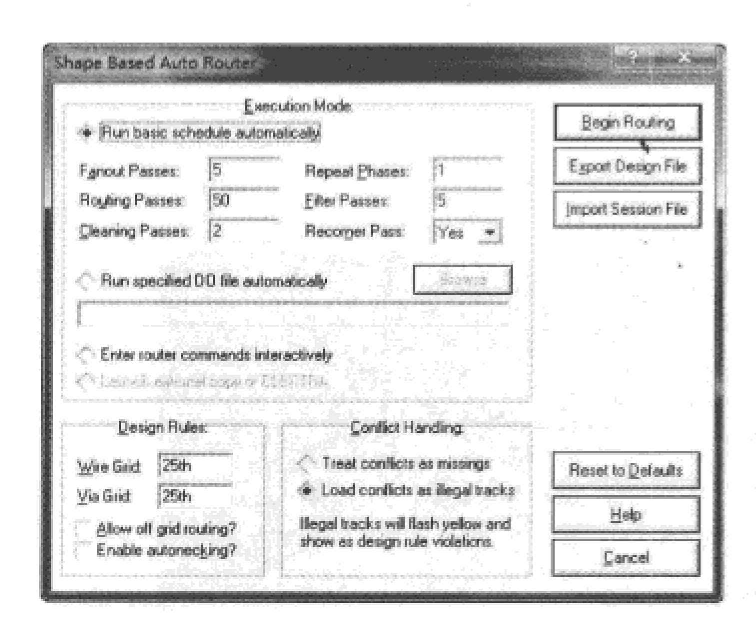


图 5.1.24　自动布线对话框

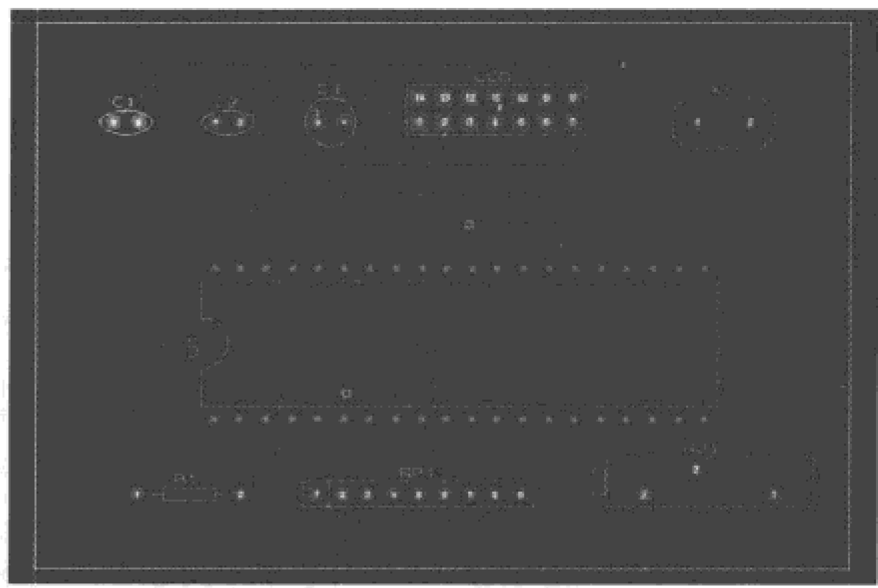


图 5.1.25　完成自动布线

(4) 布线完成后，下一步的工作是铺铜。在编辑工具栏中选择按钮，绘制出一个不超出边框的矩形，绘制方法与边框相同。绘制完成后会弹出图 5.1.26 所示的对话框，在 Net 下拉菜单中选择 VCC/VDD＝POWER，在 Layer/Colour 下拉菜单中选择 Top Copper，单击 OK 按钮确认，完成顶层的铺铜。

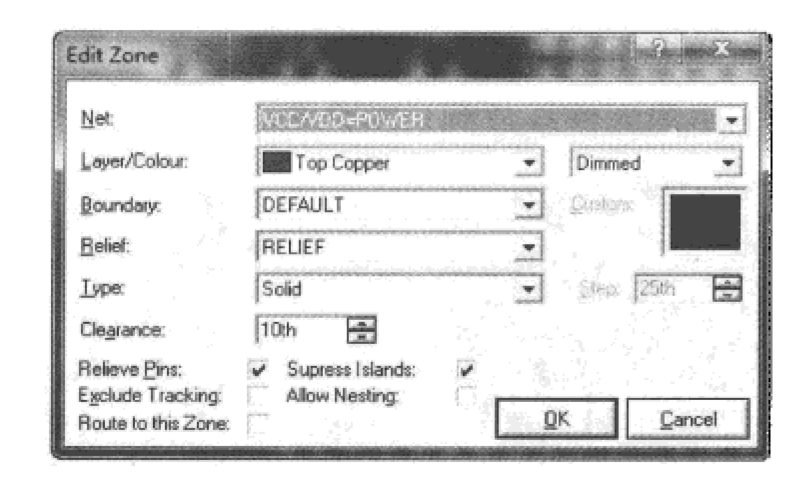


图 5.1.26　铺铜对话框

重复上面的步骤，并在 Net 下拉菜单中选择 GND＝POWER，在 Layer/Colour 下拉菜单中选择 Buttom Copper，单击 OK 按钮确认，完成底层的铺铜。如图 5.1.27 所示。

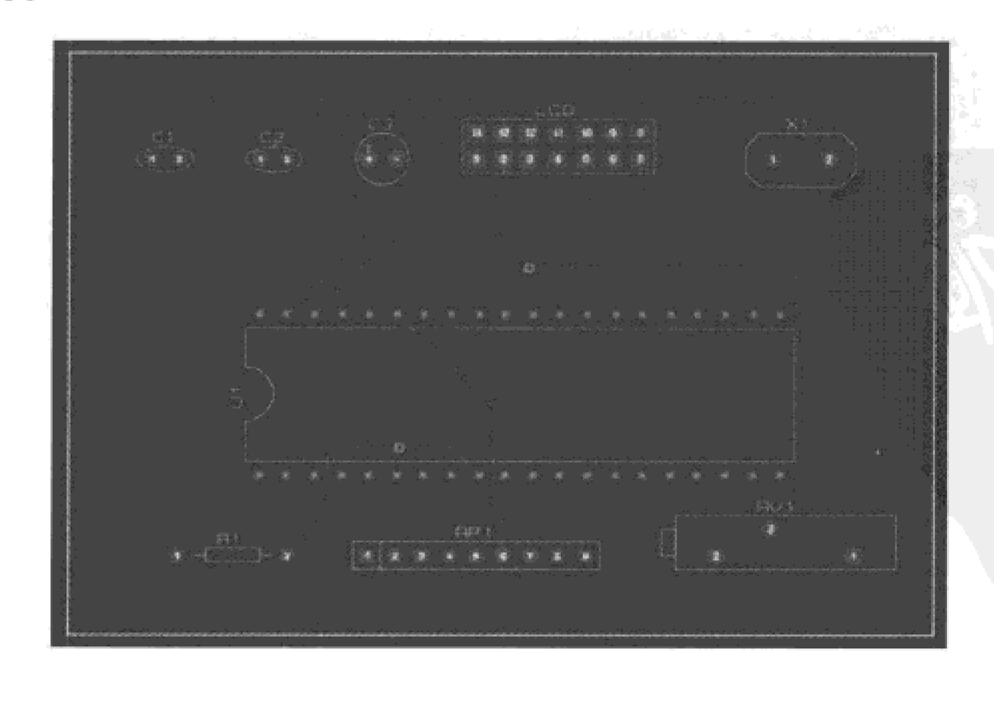
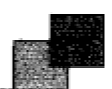


图 5.1.27　底层铺铜



(5) 完成上述步骤后,PCB制板就基本完成了,通过ARES提供的3D Visualization功能可以直接看到制板后的效果图,判断元件布局是否合理。

在菜单中选择Out put→3D Visualization即可得到图5.1.28所示的3D效果图。用鼠标或方向键可以调节多种视角进行观察,如图5.1.29所示。

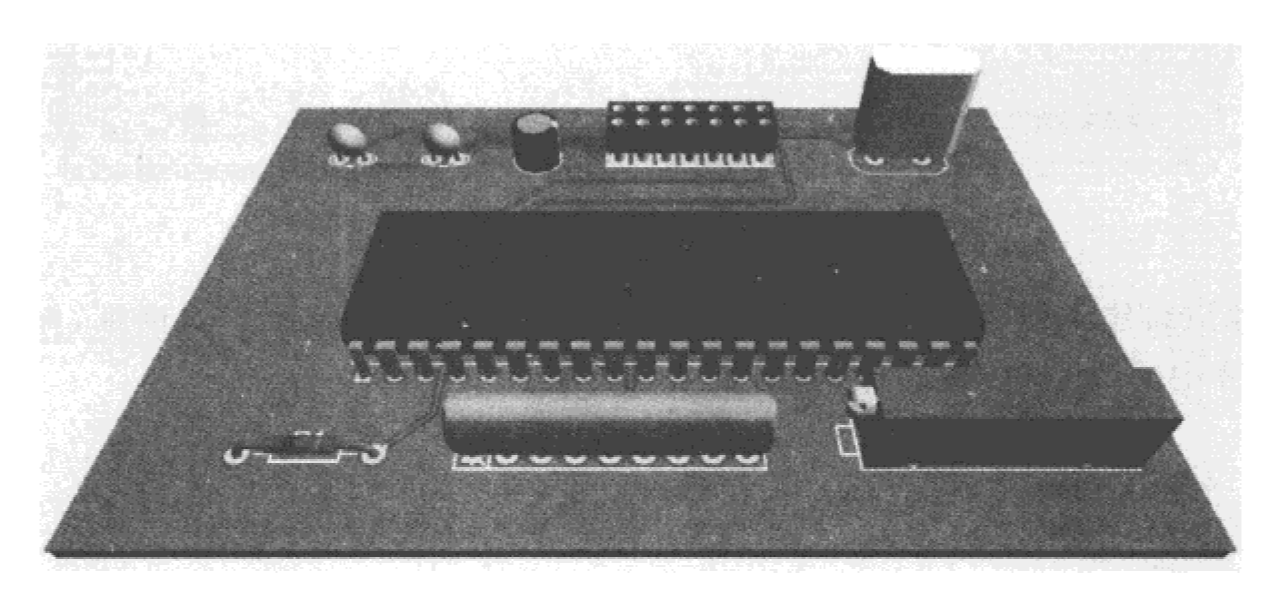


图5.1.28 PCB三维视图(一)

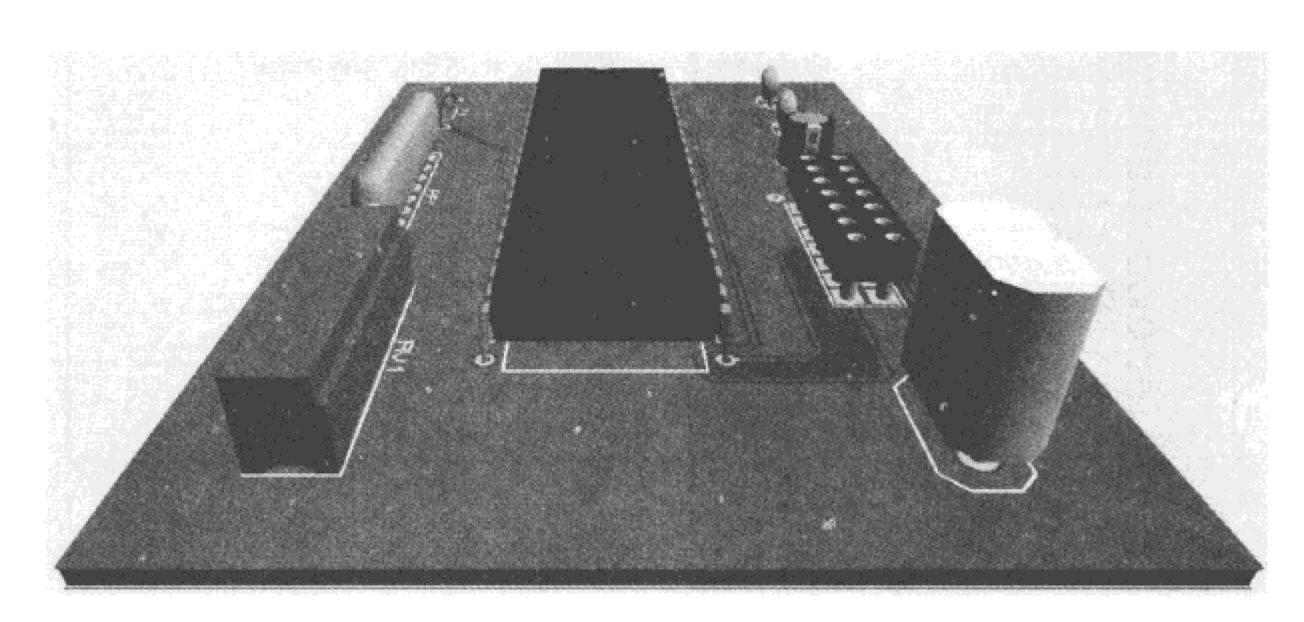


图5.1.29 PCB三维视图(二)

5.2 Keil C51集成开发环境(IDE)

5.2.1 预备知识

1. Keil C51 IDE的使用

Keil C51是一款有效的嵌入式软件开发平台,它兼容C语言和汇编语言,在国内应用相当广泛,并且Keil C51可与Proteus无缝连接,实现虚拟硬件测试,节省开发成本,提高开发效率。下面将建立一个Keil C51工程,驱动5.1.2小节建立的LCD电路。

(1) 选择菜单Project→New Project,在对话框中为工程指定保存路径、名称,如图5.2.1所示。

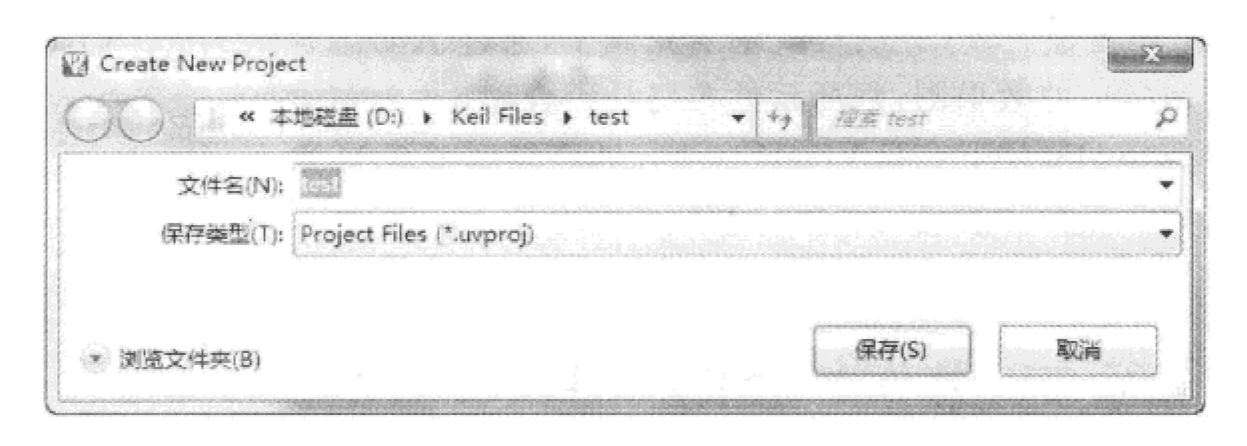


图 5.2.1　新建工程

(2) 保存该新建工程后，弹出 Select Device for Target ‘Target 1’对话框，在左侧列表框中选择 Atmel 中的 AT89C51 芯片，如图 5.2.2 所示。

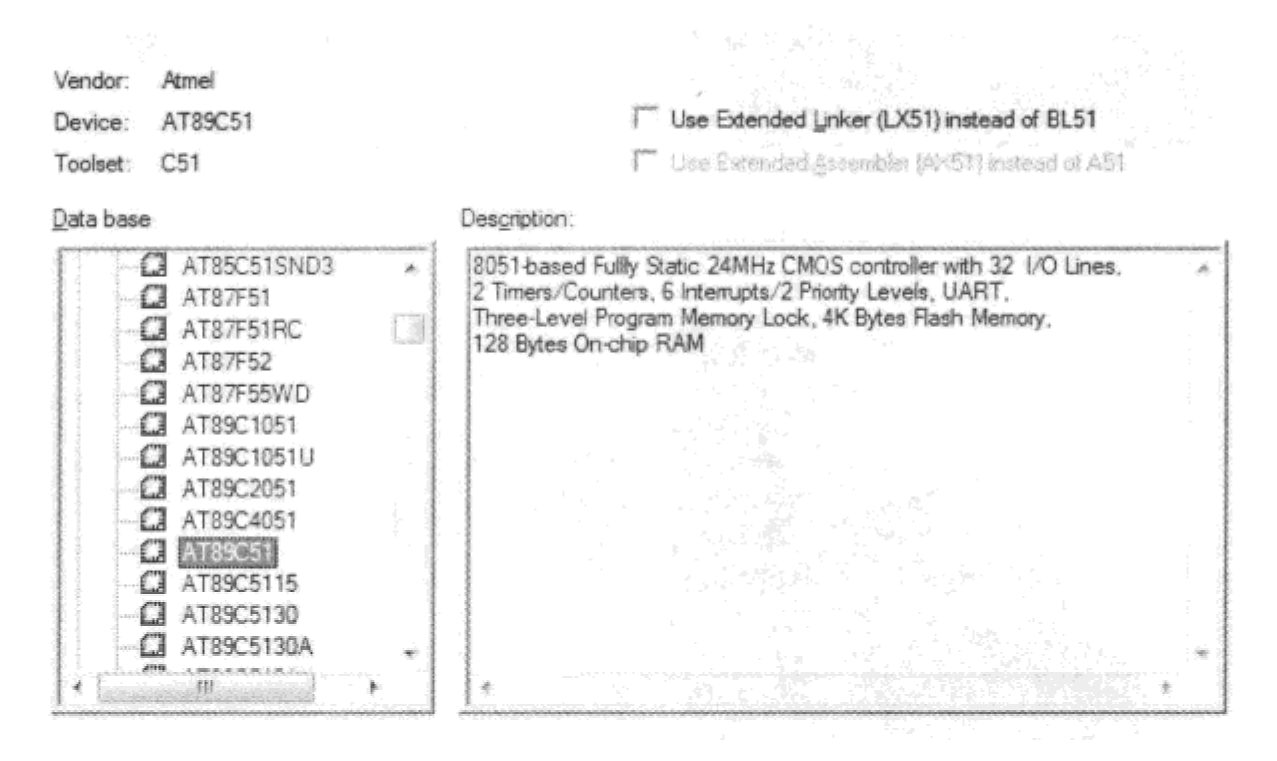


图 5.2.2　选择芯片

(3) 选择芯片并确认后，系统会提示是否将 Startup Code 复制到工程文件夹中，选择“是”，如图 5.2.3 所示。

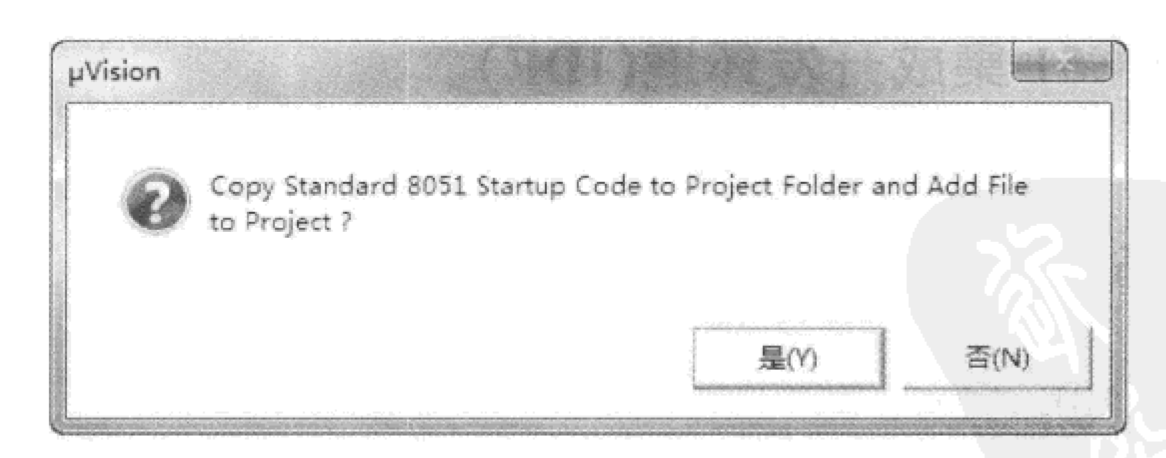
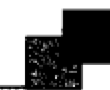


图 5.2.3　Startup Code 复制信息

(4) 工程建立完毕后，需要将源文件添加到工程中。在 Keil 界面左侧的 Project 区域，右击 Srouce Group1，选择 Add Files to ‘Source Group 1’，如图 5.2.4 所示。

在弹出对话框中，选择源文件 test.c，如图 5.2.5 所示。



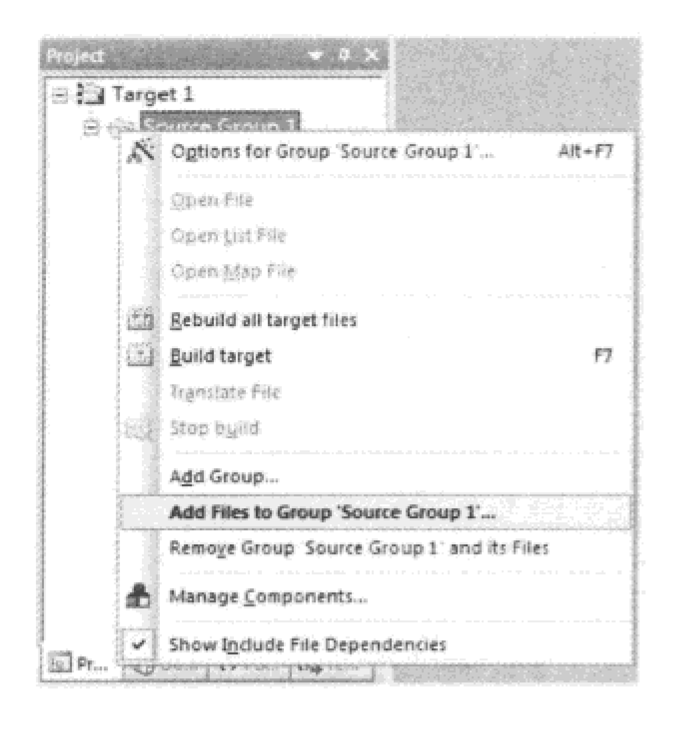


图 5.2.4　添加代码菜单

Add Files to Group 'Source Group 1'
查找范围(I): test
名称　修改日期
test.c　2010/8/4 21:46
文件名(N): test.c　Add
文件类型(T): C Source file (*.c)　Close

图 5.2.5　选择源文件

源文件内容如下：

```
#include<reg51.h>
#define uchar unsigned char
#define uint unsigned int
uchar code t1[] = "drived by 8051";                    //定义 LCD 显示的字符
uchar code t2[] = "processor! ";
sbit RS = P2^0;                                         //位声明
sbit RW = P2^1;
sbit E = P2^2;
void delay(uint x)                                      //延时函数
{  uchar y;
   while(x--)
   {
       for(y = 0;y<100;y++);
   }
}
void write_command(uchar cmd)                           //LCD 命令写入函数
{  RS  =  0;
   RW  =  0;
   E  =  0;
   P0  =  cmd;
   E  =  1;
   delay(1);
   E  =  0;
}
void write_data(uchar dat)                              //数据写入函数
{  RS  =  1;
   RW  =  0;
```

```
    E = 0;
    P0 = dat;
    E = 1;
    delay(1);
    E = 0;
}
void initialize()                                   //LCD 初始化函数
{   write_command(0x38);
    delay(1);
    write_command(0x01);
    delay(1);
    write_command(0x06);
    delay(1);
    write_command(0x0f);
    delay(1);
}
void main()
{   uint i;
    initialize();
    write_command(0x81);                            //显示第一行字符" drived by 8051"
    for(i = 0;i<14;i++)
    {   write_data(t1[i]);
        delay(200);
    }
    write_command(0x80 + 0x40 + 0x03);              //显示第二行字符"processor! "
    for(i = 0;i<10;i++)
    {   write_data(t2[i]);
        delay(200);
    }
    while(1);
}
```

(5) 文件添加成功后，单击界面上方工具按钮，进行简单配置。

在 Target 面板中，将晶振频率设置为 12MHz。但如果是硬件，此处不需修改，程序会以硬件实际频率运行，如图 5.2.6 所示。

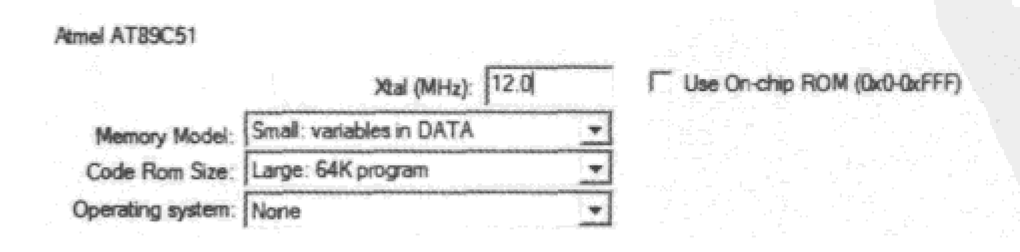
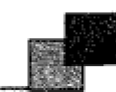


图 5.2.6　设置晶振频率

在 Output 面板中选中 Create HEX File，在编译时将生成 HEX 文件，可以在 Proteus 中



仿真，也可以下载到硬件中运行，如图 5.2.7 所示。

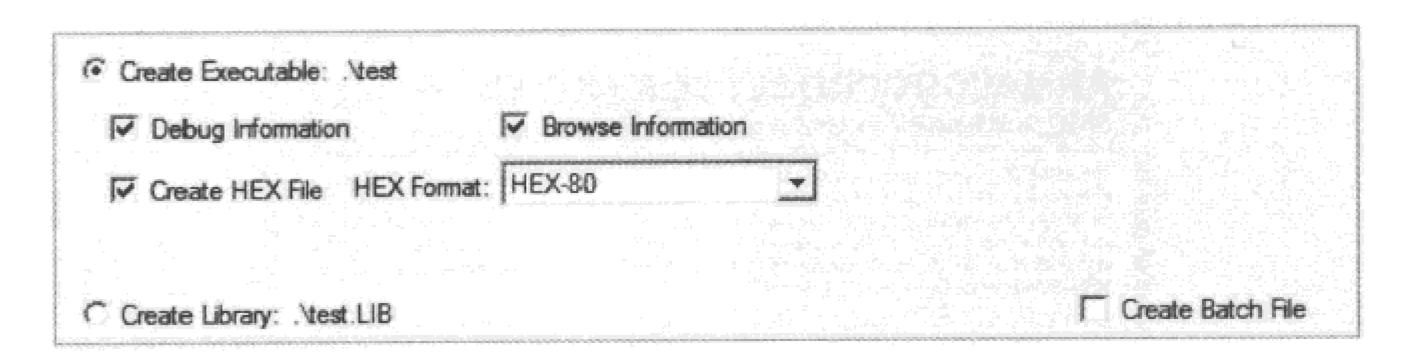


图 5.2.7　设置输出文件格式

(6) 配置完成后，单击快捷按钮开始编译并连接工程，如图 5.2.8 所示。

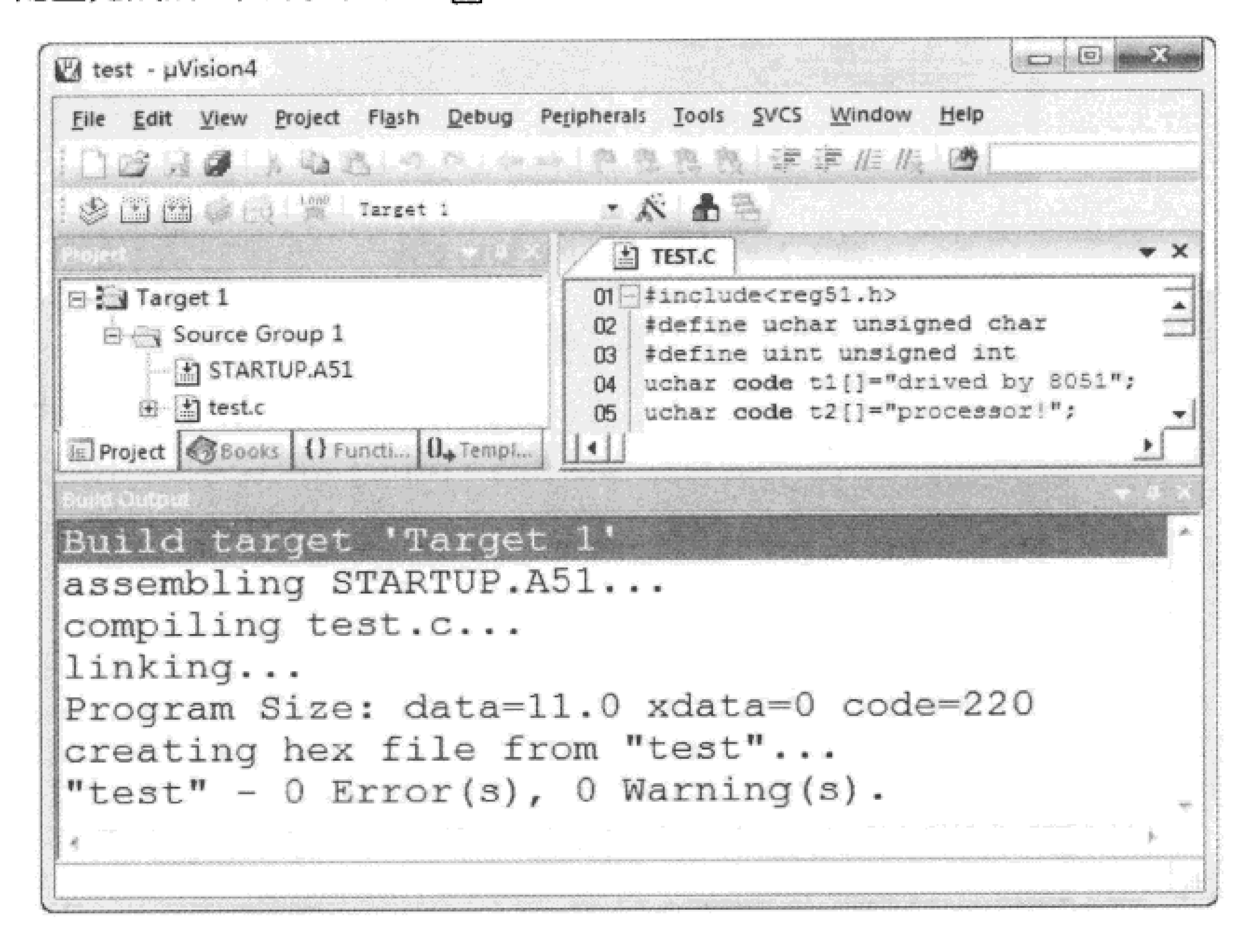


图 5.2.8　编译工程

2. Keil C51 与 Proteus 的联调

安装 Keil C51 V9.01 版与 Proteus 7.7 版后，还需要进行简单的配置才能将其整合。

(1) 打开 proteus 安装目录的帮助文件夹，如 x:\Program Files\Labcenter Electronics\Proteus 7 Professional\HELP，找到帮助文件 ARM.HLP，在首页单击链接 Remote Debugger Drivers(图 5.2.9)。

在 Remote Debugger Drivers 界面，选择链接 Download and Install remote debugger driver for Keil uVision3(图 5.2.10)。

随后将联调插件 vdmagdi.exe，保存在本地磁盘(图 5.2.11)。

(2) 运行 vdmagdi.exe 后，按照安装向导一步步完成安装，如图 5.2.12～图 5.2.16 所示。

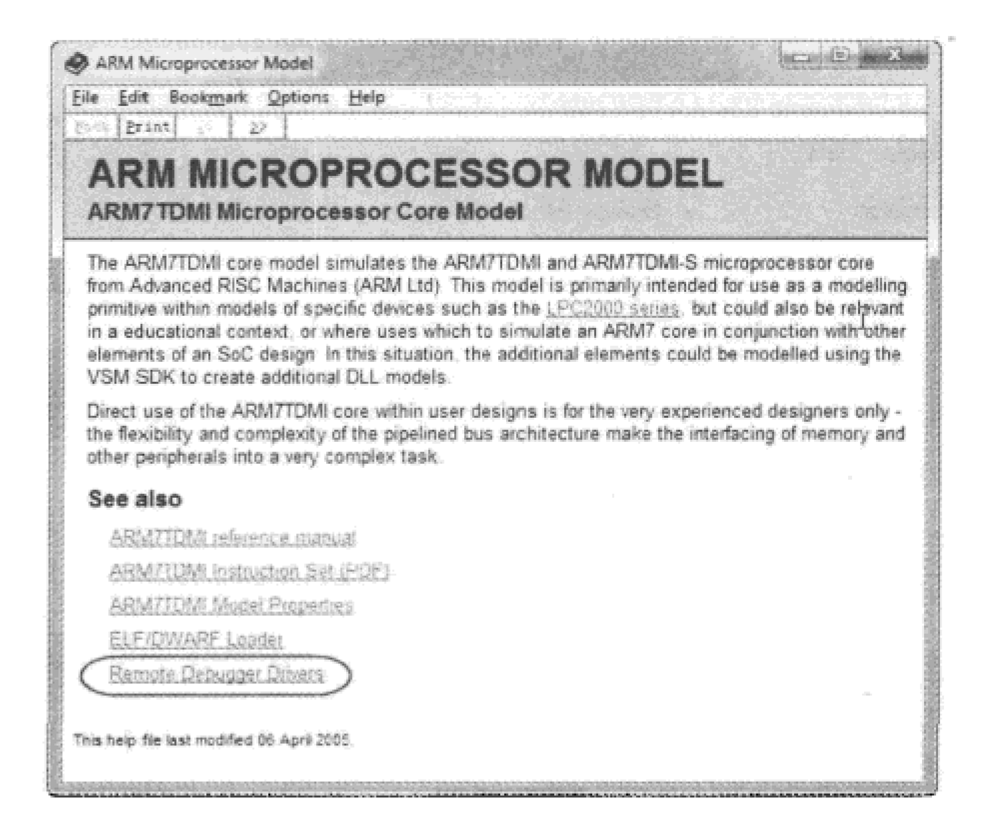


图 5.2.9　Proteus ARM 帮助文档

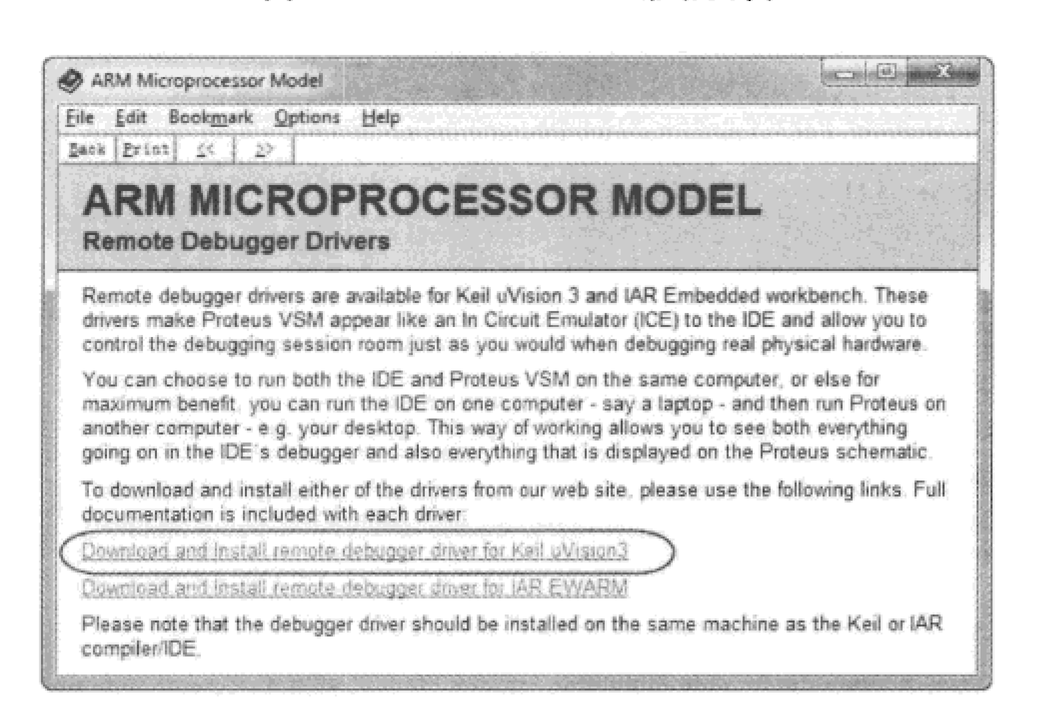


图 5.2.10　Remote Debugger Drivers 界面



图 5.2.11　保存联调插件

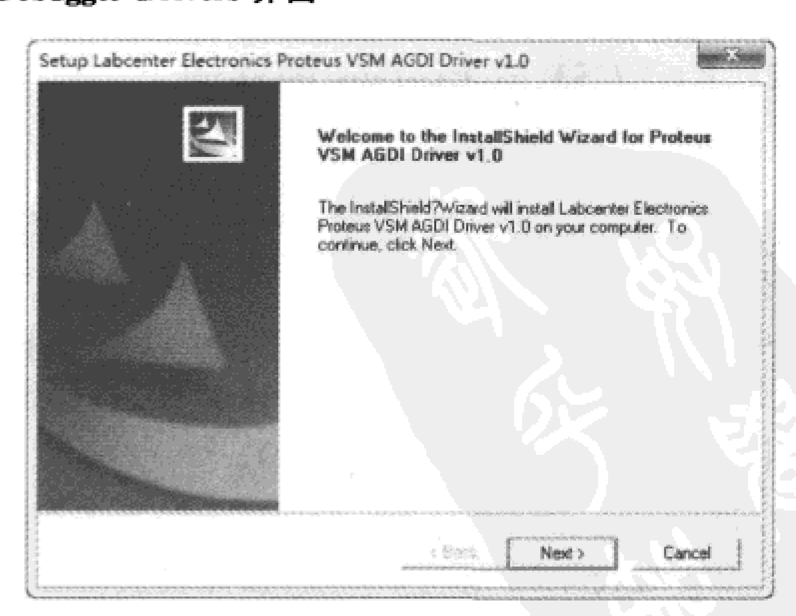
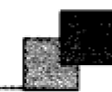


图 5.2.12　安装起始界面



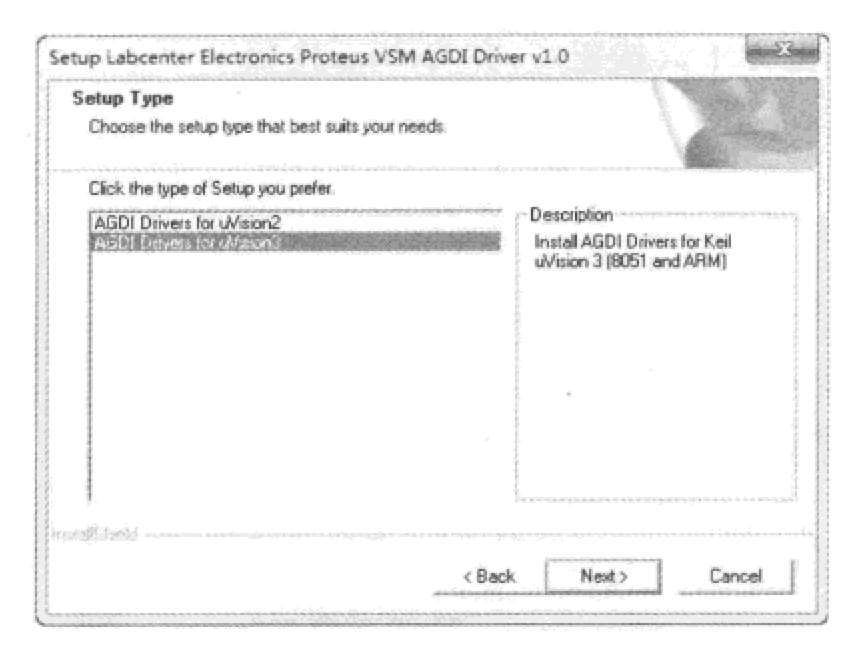


图 5.2.13　选择 Keil 版本

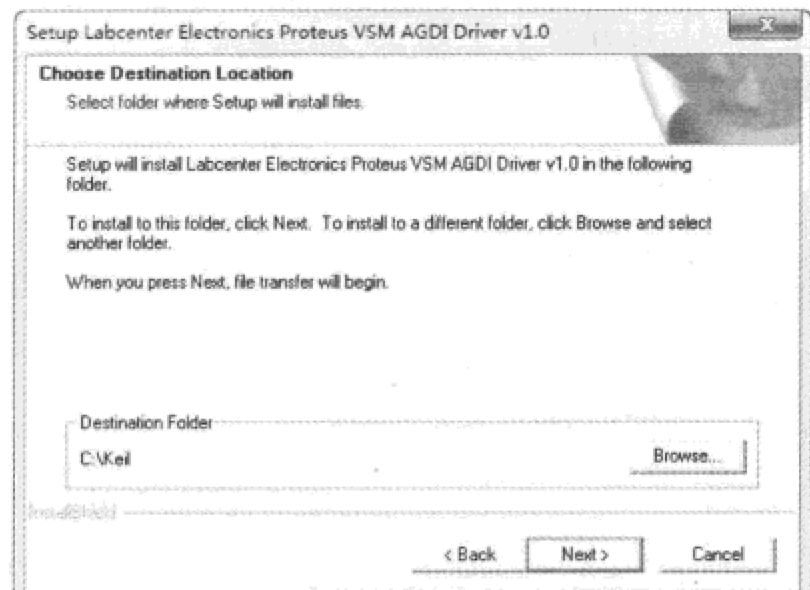


图 5.2.14　选择路径

在组件选择对话框中同时选中 8051 AGDI Driver 和 ARM AGDI Driver，以便 ARM 工程也可以和 Proteus 进行联调(图 5.2.15)。

安装成功如图 5.2.16 所示。

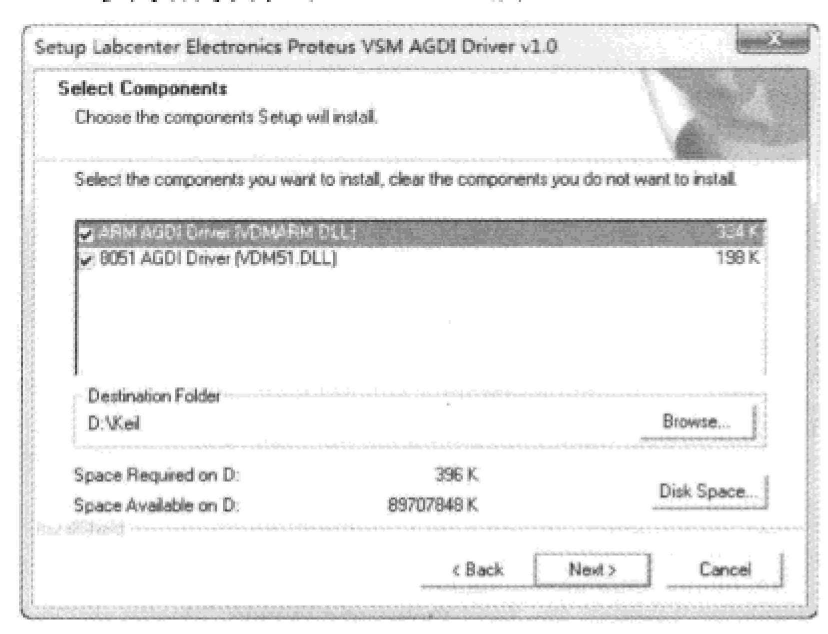


图 5.2.15　组件选择

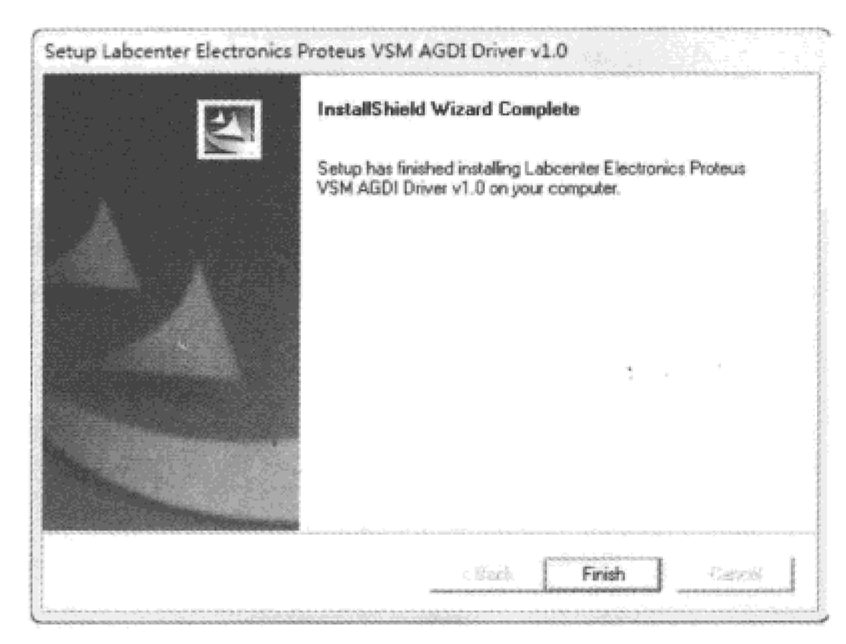


图 5.2.16　安装完成

(3) 打开第 5.1 节完成的 Proteus 原理图，在菜单栏中选择 Debug→Use Remote Debug Monitor。如图 5.2.17 所示。

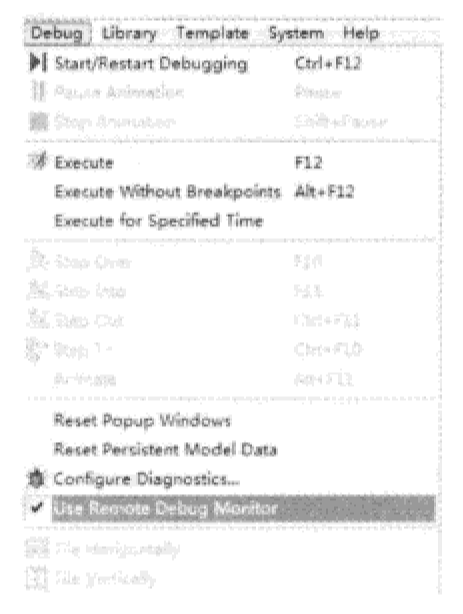


图 5.2.17　启用远程调试



(4) 打开上文中建立的 Keil 工程，单击界面上方工具按钮，在 Debug 选项卡中选择使用 Proteus 仿真器，如图 5.2.18 所示。

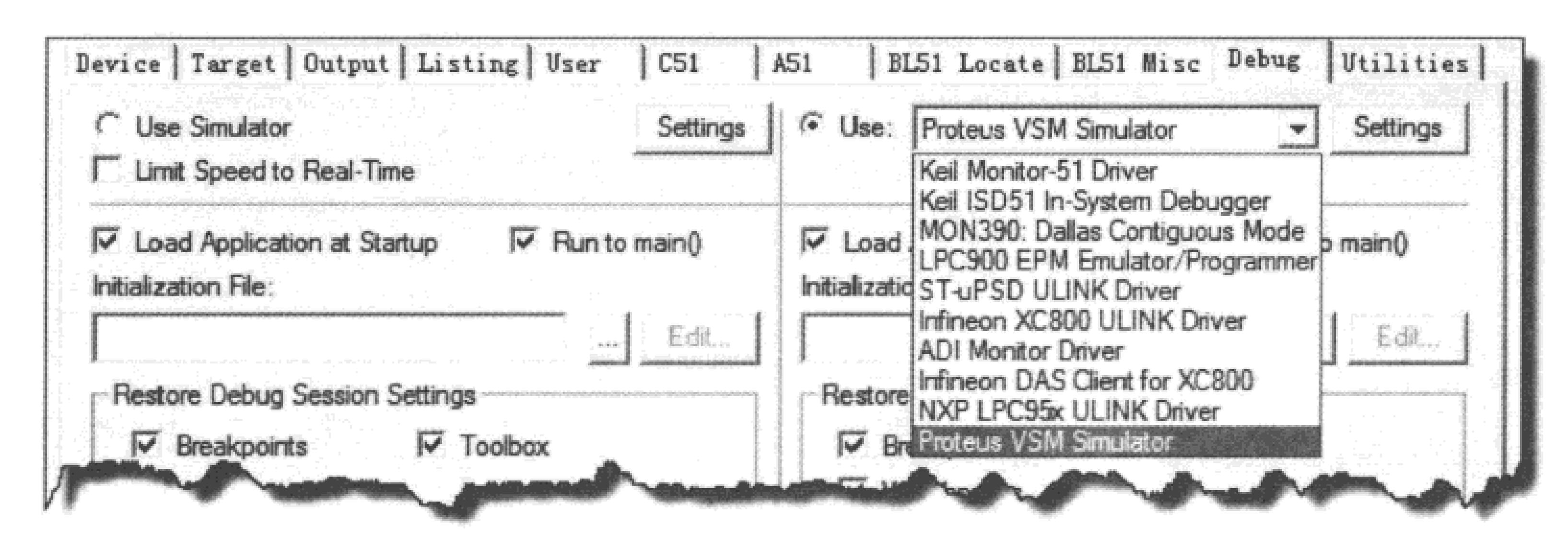


图 5.2.18 选择 Proteus 仿真器

至此，Keil 和 Proteus 已经完成了整合，可以实现联调功能了。在 Keil 中单击按钮开始调试。如图 5.2.19 所示。

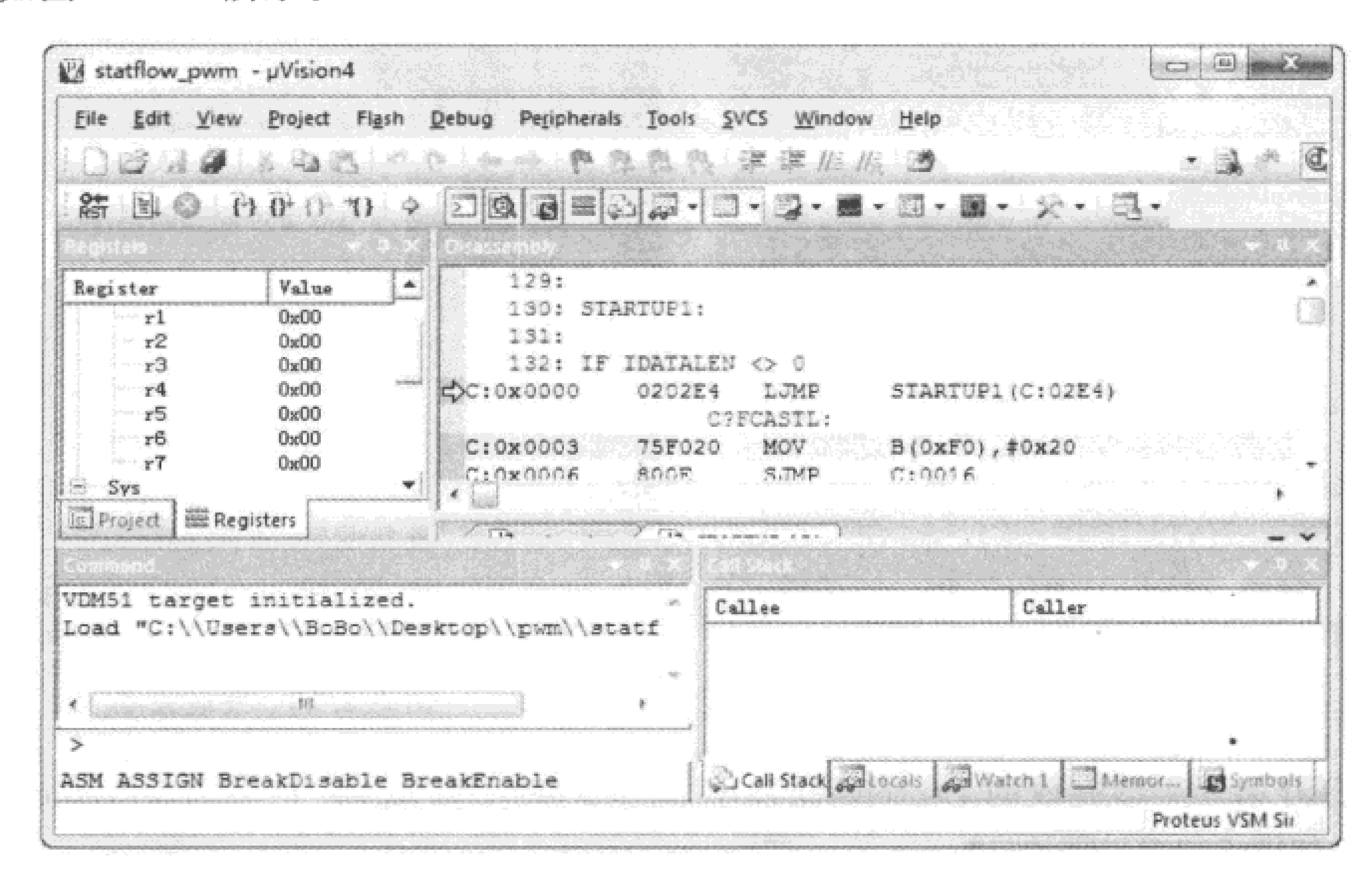


图 5.2.19 Keil 调试界面

单击全速运行按钮，Proteus 中的原理图会同步显示运行结果，如图 5.2.20 所示。

除了全速运行按钮外，调试环境下还提供了：

① 程序复位。

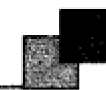
② 停止全速运行。

③ 进入子函数。

④ 单步运行。

⑤ 跳出子函数体。

⑥ 运行至光标所在行。



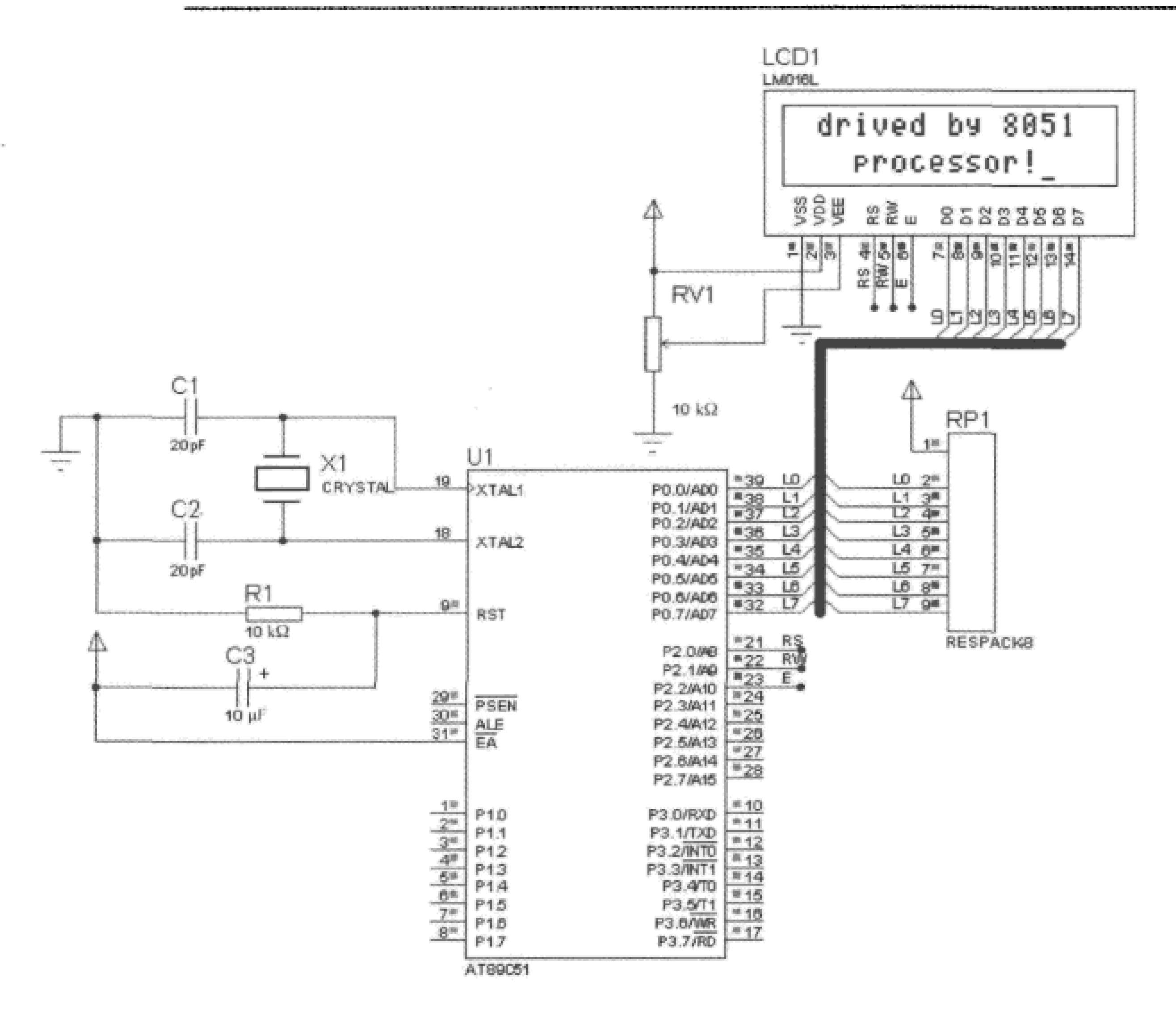


图 5.2.20　Proteus 运行结果

在调试模式中利用以上功能按钮，可以实现单步、全速、断点调试。

设置断点：调试过程中，用户有时能确定错误在程序中大概的位置，可以通过设置断点，将程序快速运行到关键位置。

在期望设置断点的程序行序号左侧双击，即可设置断点。当单击全速运行按钮后，程序将迅速定位到断点所在位置，如图 5.2.21 所示。

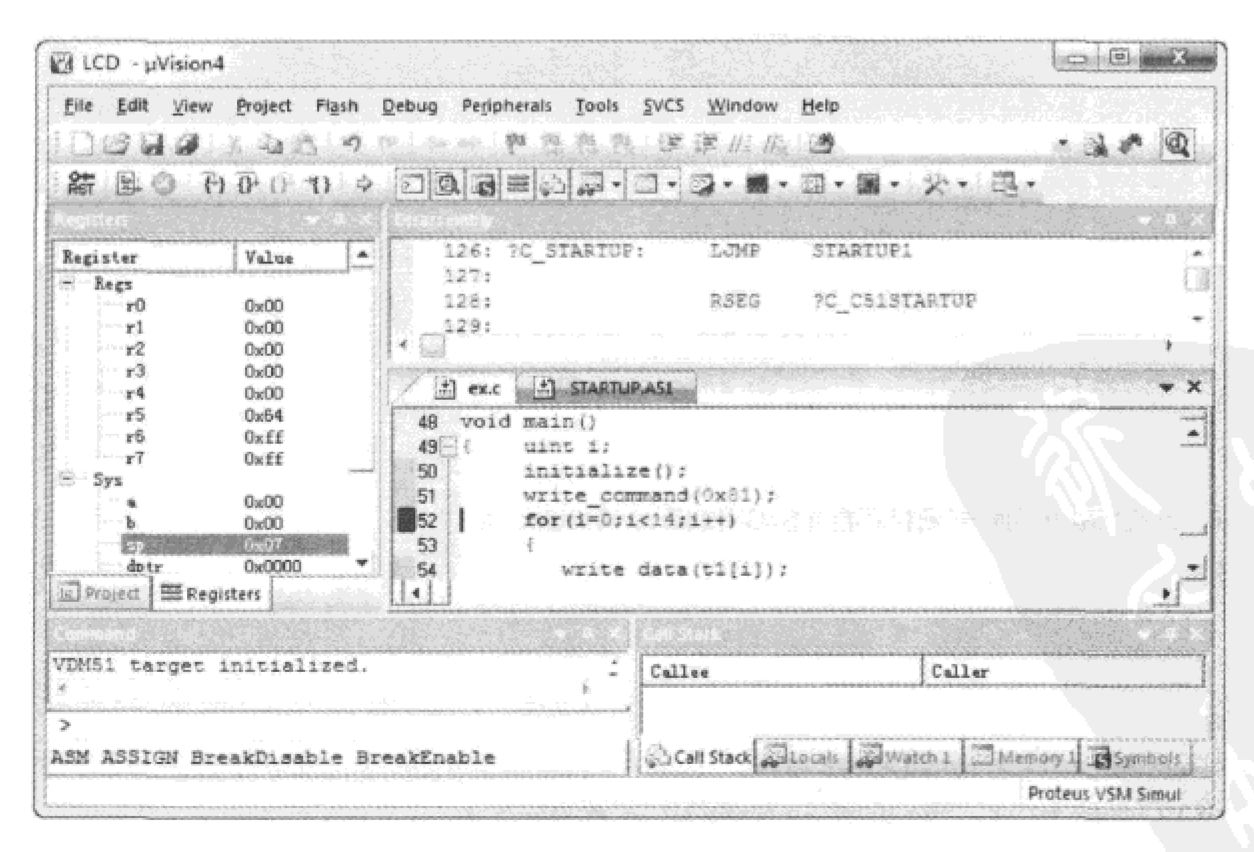


图 5.2.21　设置断点

观察变量的值：对于一些重要的变量，需要一直关注其变化。单击右下方的 watch1 子选项卡，在 Name 区域下方双击或按 F2 键，即可设置需要观察的变量。例如，输入程序中的变量 i，如图 5.2.22 所示。

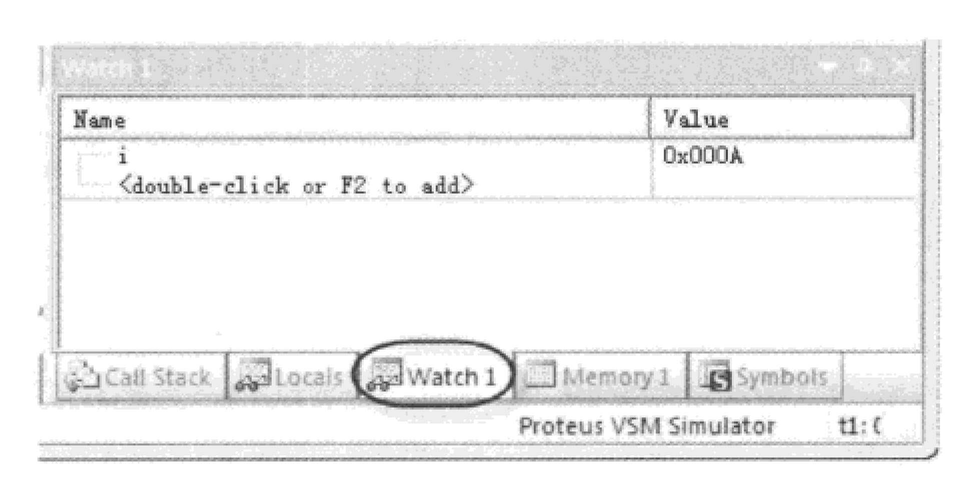


图 5.2.22　观察变量值

观察 I/O 口的值：对于本例的单片机实验，P0 口的电平决定了 LCD 上的显示结果，通过菜单栏上的 Peripherals→I/O－Ports→Port 0 可以调出 P0 口状态，如图 5.2.23 所示。

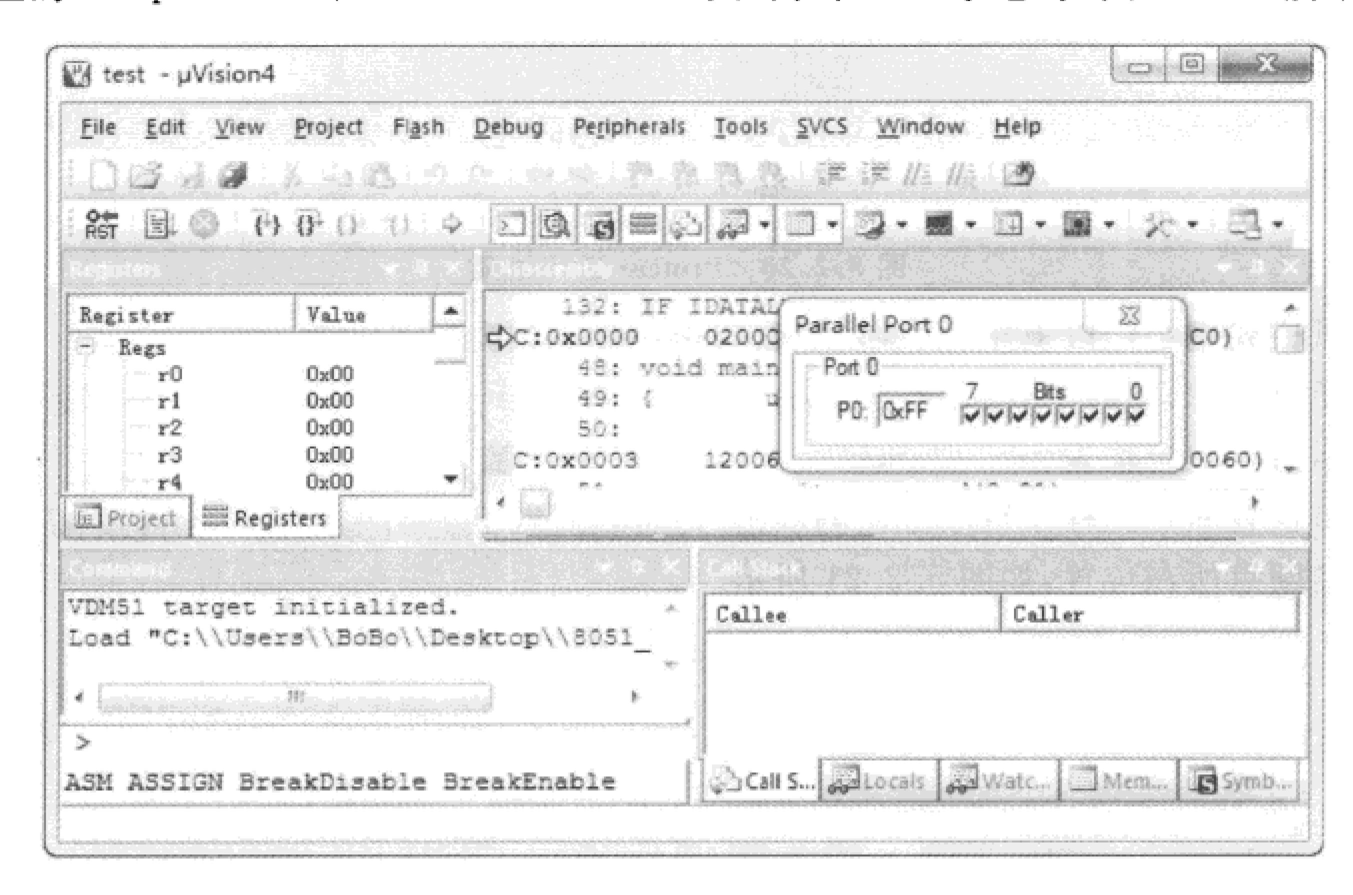


图 5.2.23　观察 I/O 接口值

在 Proteus 界面中，同样提供了观察芯片寄存器的工具。在单步调试时，右击 AT89C51 芯片，选择 8051 CPU 子菜单，可以看到，Registers、Internal(IDATA) Memory、SFR Memory 子项，选中任意一个即可显示相应存储器的内部数据，如图 5.2.24 所示。

例如选择 Registers，弹出图 5.2.25 所示的窗口，可以看到当前机器指令及各寄存器的数值。

若选择 SFR Memory，则弹出图 5.2.26 所示的窗口，该窗口中的值是与实际物理地址相对应的。例如 P1 对应的实际物理地址为 0x90(具体可查看 8051 头文件)，其中的数据为 FF，与 Registers 窗口的值相符。发生变化的值以高亮形式表示。



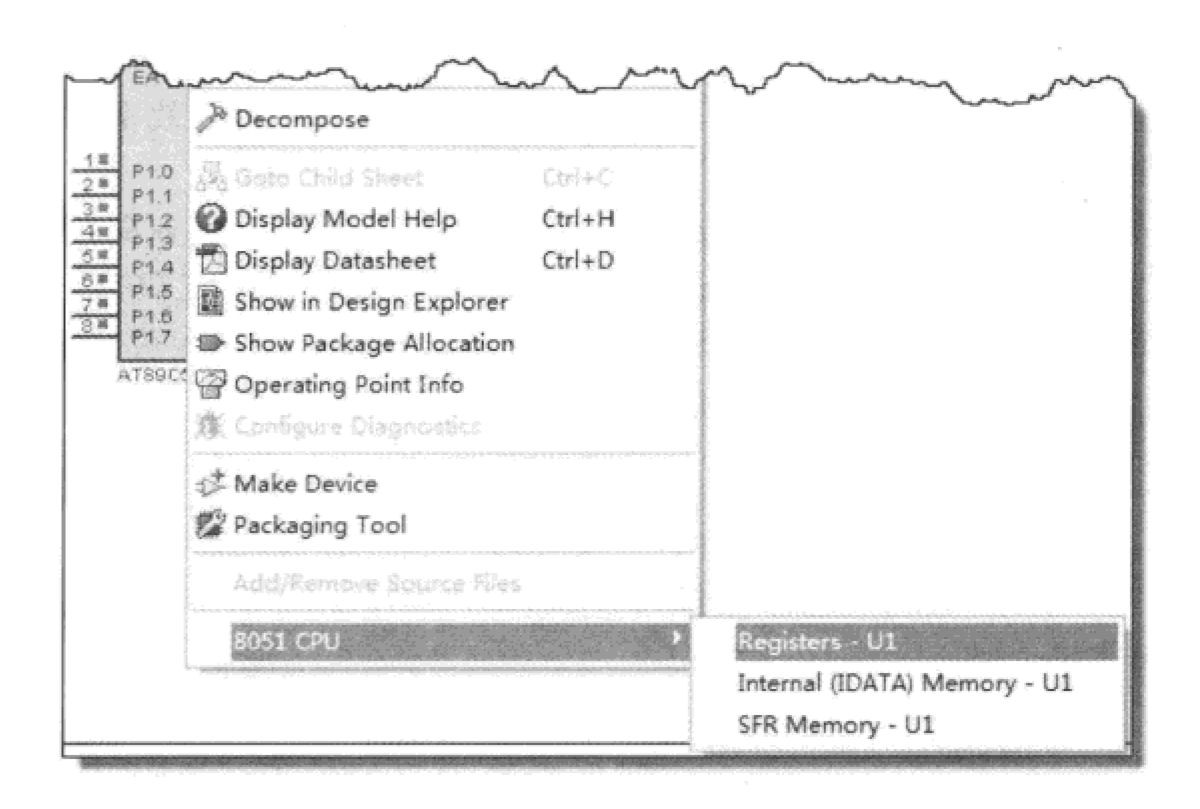


图 5.2.24　8051CPU 芯片菜单

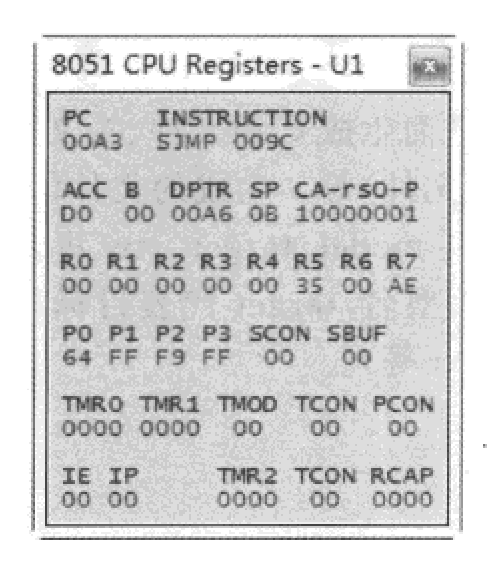


图 5.2.25　寄存器观察窗

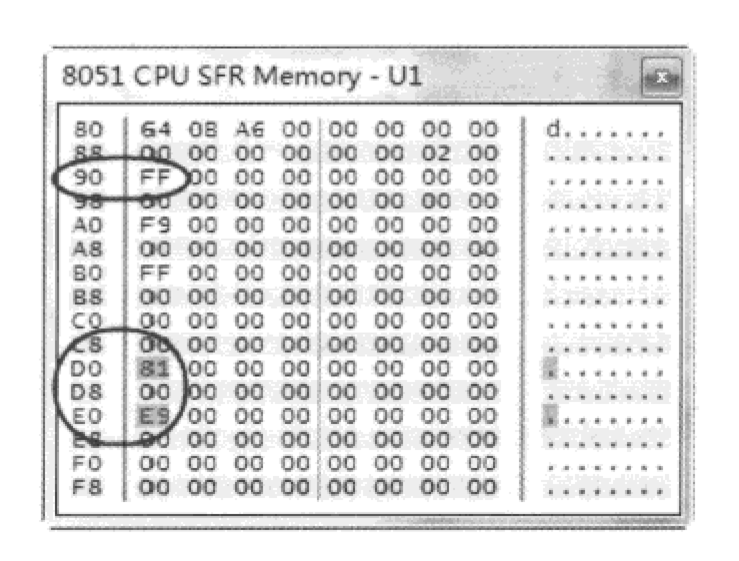


图 5.2.26　SFR 存储器观察窗

5.2.2　RTW-EC 快速代码生成

Real-Time Workshop Embedde Coder(RTW-EC)是 Real-Time Workshop(RTW)自动代码生成工具的扩展，它加入了多种对于嵌入式软件开发至关重要的功能。用户使用 RTW-EC，可以得到清晰、紧凑、高效、接近专家手写的 C 代码，这对于嵌入式系统或大型设备的实时仿真、快速原型建立是相当必要的，还可以自行定义生成代码的形式，针对特定的目标环境进行代码优化，继承或集成现有的函数与数据，建立代码与模块之间的关联，进行代码验证。其主要特点如下：

(1) 从 Simulink 和 Stateflow 模型中生成 ANSI/ISO C 和 C++代码及其可执行文件，生成的代码在内存占用率、运行速度及可读性等方面可同手写代码相媲美。

(2) 扩展了 Real-Time Workshop 和 Stateflow Coder，其在产品实现方面具有最优化及代码配置等特点。

(3) 支持所有 Simulink 数据对象和数据字典功能，包括用户定义的存储类、类型及别名。

(4) 提供目标函数库代码的定制,从而为特定处理器生成机器代码。

(5) 无论有无 RTOS,均可对多速率代码进行简明分割以提高运行效率。

(6) 包含可扩展的模块封装特性和自定义数据对象。

(7) 提供详尽的注释,并使用超链接进行代码到模型和需求之间的双向跟踪。

(8) 自动将生成的代码导入 Simulink 进行软件环路测试,从而对代码进行验证。

(9) 使用 Simulink 报告生成功能在 Simulink Model Explorer 中生成代码帮助文件并以此作为独立的报告。

1. 代码生成原理

作为预备知识,这里简要列出 RTW-EC 生成代码的原理及各代码文件的意义,读者宜在完整阅读了后续各节并实际操作,有了感性认识后,再返回理解本小节。关于 RTW-EC 更详细的说明,请参考帮助文档。

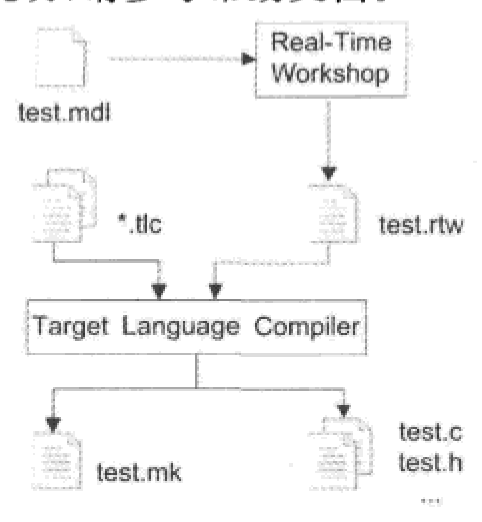


图 5.2.27　C 代码生成过程

图 5.2.27 是 TLC 目标语言编译器与 RTW 代码生成器生成 C 代码的过程。首先,RTW 将 Simulink 模型生成对应的 RTW 文件,该文件包括了生成代码所需要的信息,这些信息应由用户在建模时指定。然后 TLC 编译器读取 RTW 文件,据此选择系统 TLC 和模块 TLC 文件,进而生成 C 源文件、头文件、MK 文件(用于生成可执行文件)等。生成的文件保存在与模型同目录下的文件夹 model_target_rtw 里,名称中的 model 代表实际的模型名,target 代表目标环境(实际操作时可能是 ert、grt 等)。

2. 建模及代码生成

图 5.2.28 是一个简单的放大模型,模型名为 test.mdl。

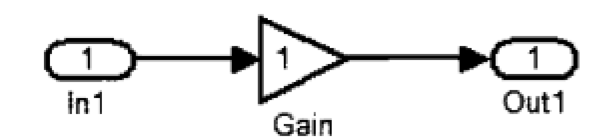


图 5.2.28　放大模型 test.mdl

选择模型主窗口的菜单项 Simulation→Configuration Parameters…(图 5.2.29),打开模型参数对话框(图 5.2.30)。

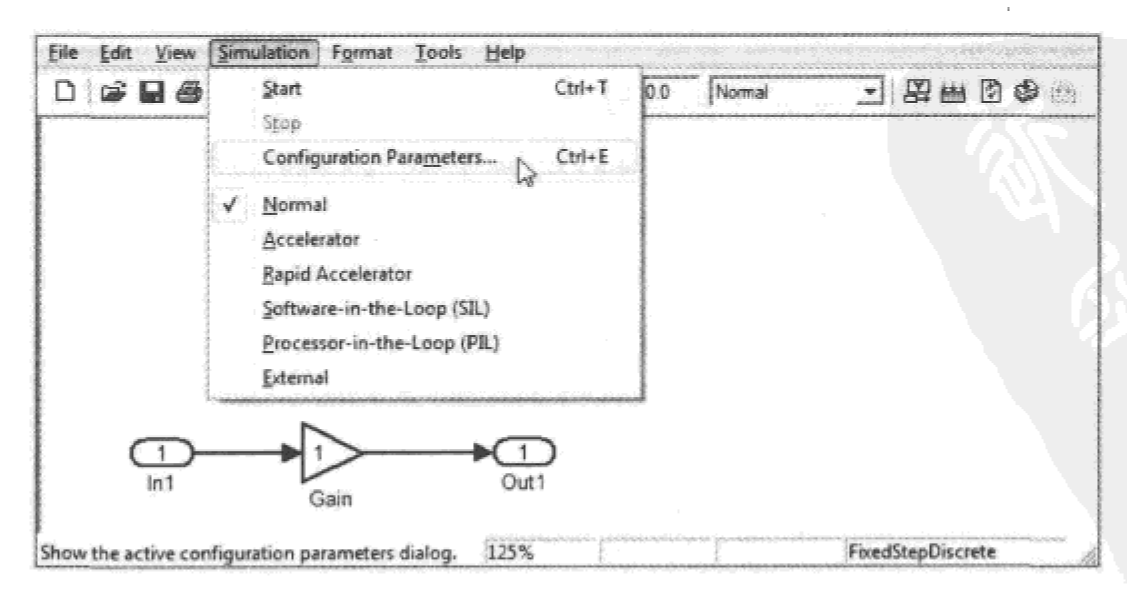
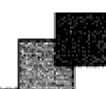


图 5.2.29　模型窗口



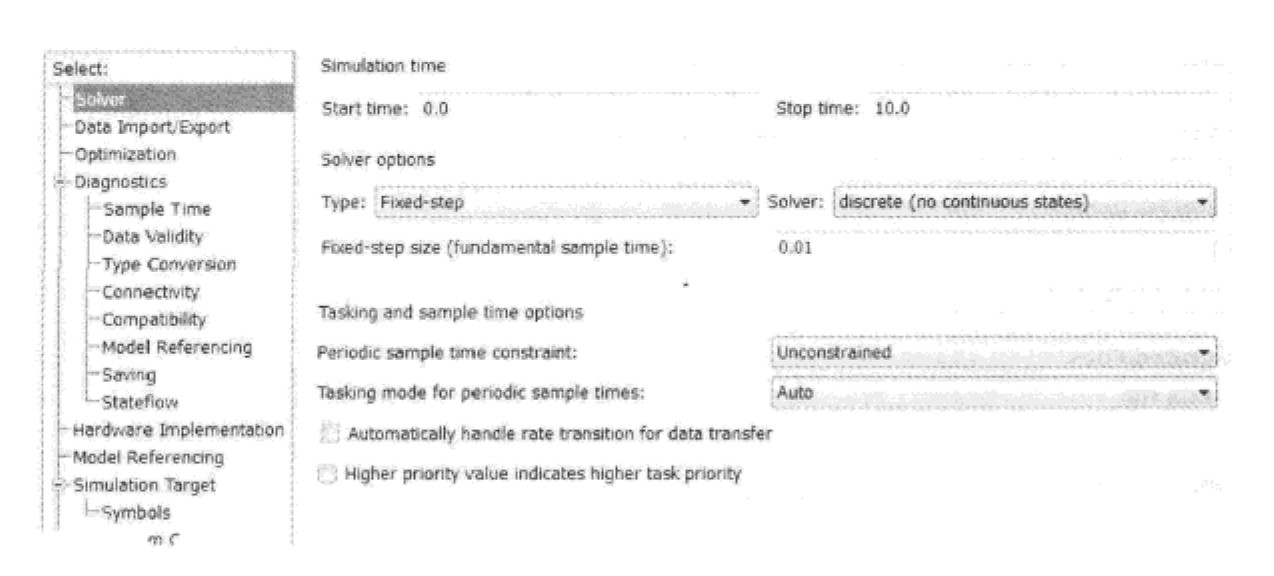


图 5.2.30　参数设置对话框

在 Solver 面板，设置求解器为定步长离散求解器，步长为 0.01(图 5.2.31)。

Real-Time Workshop 面板，设置 TLC 文件为 ert.tlc(图 5.2.32)。

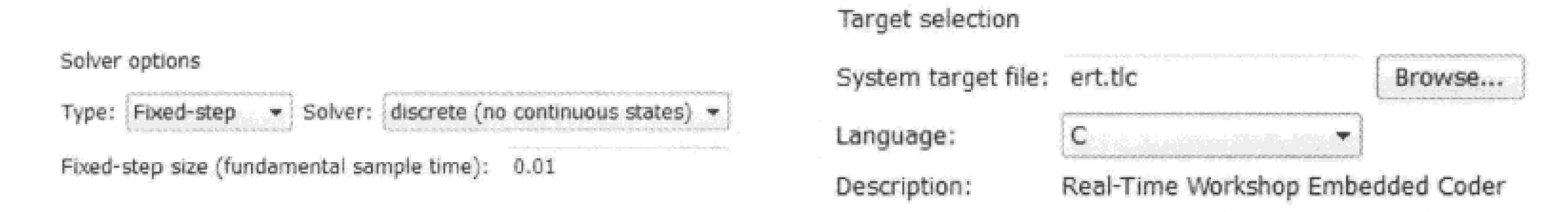


图 5.2.31　选择求解器　　　　图 5.2.32　选择 TLC 文件

Real-Time Workshop→Report 面板，勾选所有复选框，便于后期检查及跟踪(图 5.2.33)。

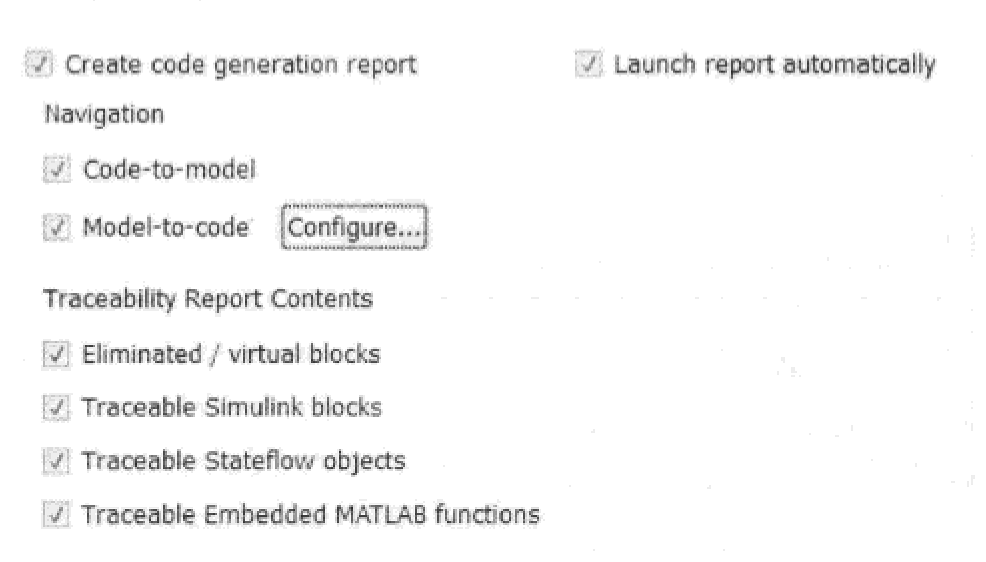


图 5.2.33　报告生成设置

单击模型工具栏的按钮(图 5.2.34)，生成代码，报告如图 5.2.35 所示。

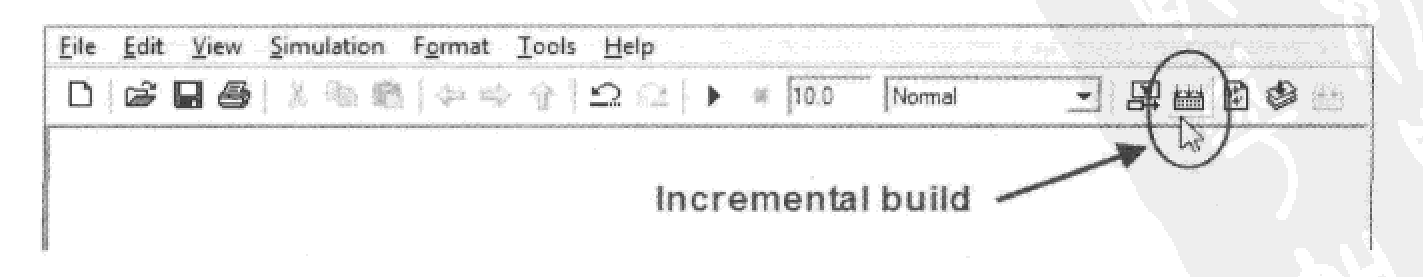


图 5.2.34　生成代码

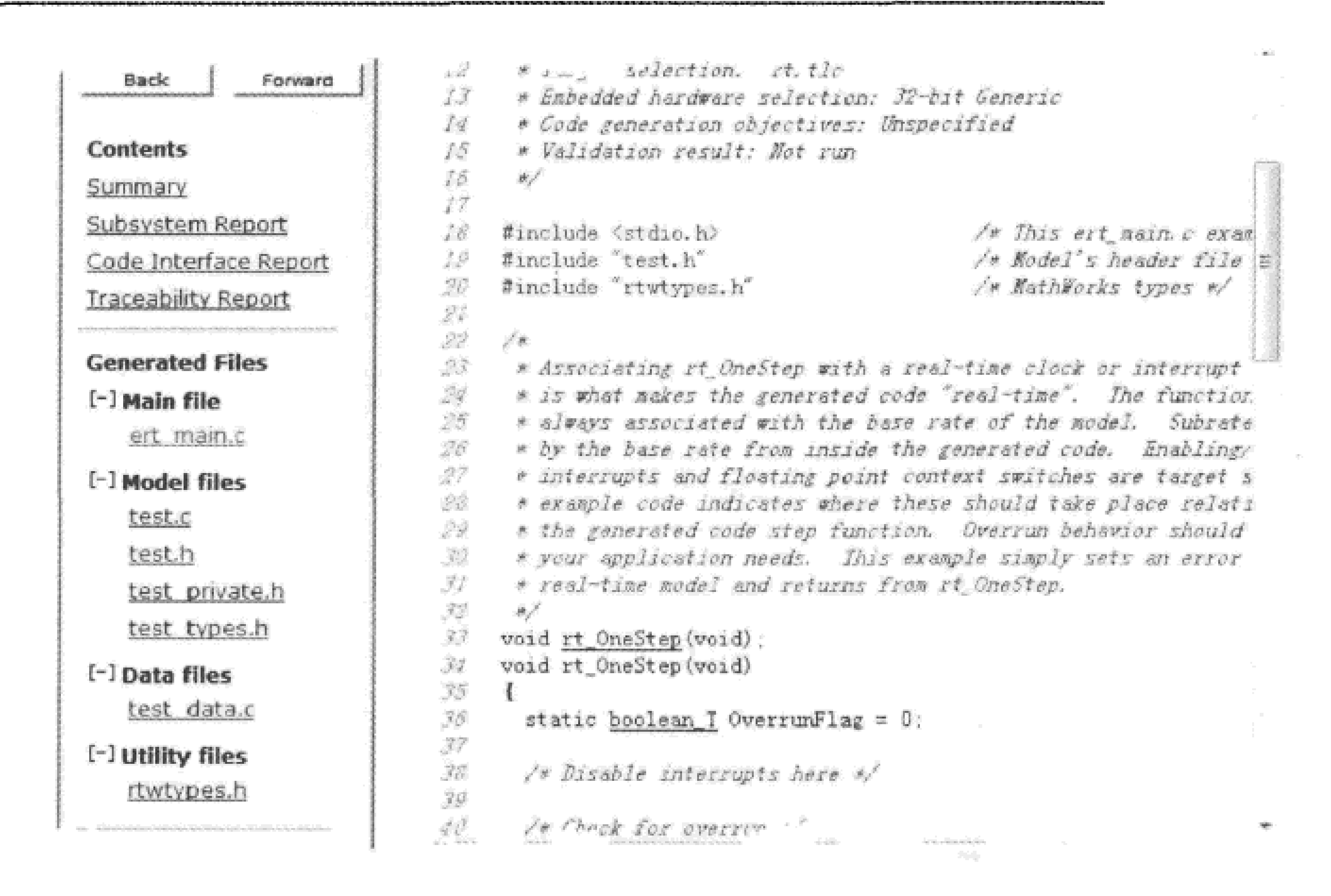


图 5.2.35　代码报告

3. 代码文件分析

读者根据报告上的文件分类，大致能了解这些代码的功能。而对于不同的求解器、不同的 TLC 文件（如第 8 章）以及模型的其他优化设置，生成的代码则不尽相同，但整体的代码文件结构不会有太大差异。因此本节以定步长离散求解器与 ert.tlc 为基础，说明代码的意义，其中各文件名中的 model 即表示模型名。

(1) ert_main.c。ert_main.c 只是一个样板文件，虽然它包含了 C 语言最主要的 main 函数，并在 main 函数中调用了 model_initialize、model_step、model_terminate 等模型函数，但除此之外并没有进行其他更多的操作。因此多数情况下，用户应根据实际需要在 ert_main.c 中添加必要的与硬件有关的代码，例如头文件、中断服务程序、硬件初始化代码、算法与硬件接口代码、循环语句等。当然用户也可以自行编写一个 main.c 文件，调用各模型函数。

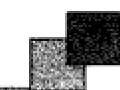
(2) model.c。该文件包含了模型函数 model_initialize、model_step、model_terminate 的入口地址及其代码，这些代码实现了整个模型的算法，其中函数 model_step 实现了模型中所有模块的功能。

本例的 test_step 函数实现了输入 In、增益、输出 Out 三个模块的功能。

(3) model.h。该文件声明了模型的数据结构以及模型输入与数据结构间的全局接口，并将模型函数 model_initialize、model_step、model_terminate 声明为外部函数。model.c 与 model_data.c 将引用该头文件。

(4) model_private.h。该文件包含了模型与子系统需要用到的本地宏与本地数据，如果模型中含有输入自外部模块的信号，则该文件还将体现这些信号的配置常量与声明，model.c 与 model_data.c 将引用该头文件。

(5) model_types.h。该文件提供了实时模型数据结构与参数结构的前向声明，可重用函数的函数声明可能需要这些声明，同时该文件也定义了用户在模型里自定义的类型，model.c 将引用该头文件。



(6) model_data.c。该文件声明了参数数据结构、常数模块数据结构以及模型中所有零值表示的数据类型。不过如果模型没用到这些数据类型与零值表示,则不生成该文件。

模型 test.mdl 含有一个带参数的增益模块,因此 test_data.c 包含了该增益模块的参数:放大倍数,model.c 将引用该头文件。

(7) rtwtypes.h。该文件定义了由 RTW-EC 生成代码所用到的数据类型、结构体与宏,所有有用到这些数据类型的文件,都应引用该头文件。

上述几个文件之间的引用关系,可以用图 5.2.36 表示,图中箭头指向的文件引用了箭头源的文件。

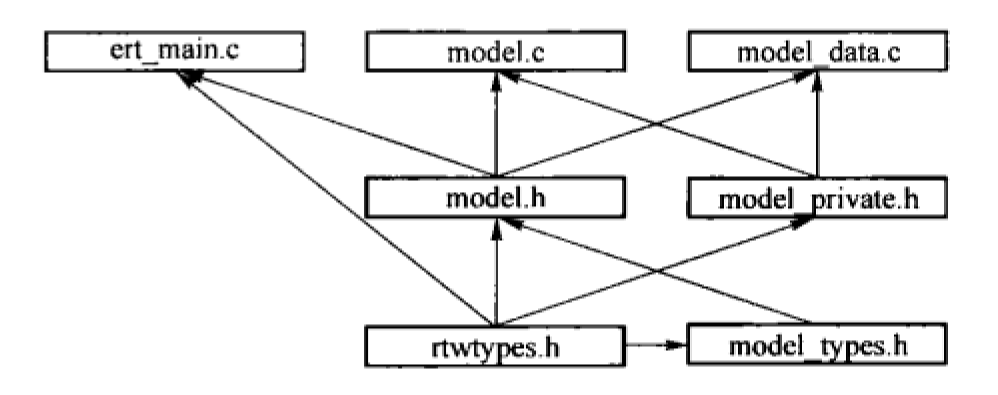


图 5.2.36 文件引用关系

在模型参数设置对话框的 Real-Time Workshop→Templates 页面,选项 Generate an example main program 可控制是否生成 ert_main.c 文件,默认情况选中(图 5.2.37)。



图 5.2.37 选择是否生成 ert_main.c 文件

在模型参数设置对话框的 Real-Time Workshop→Code Placement 页面,选项 File packageing format 可控制其余文件的生成情况(图 5.2.38),表 5.2.1 列出了不同格式时代码文件的生成情况。

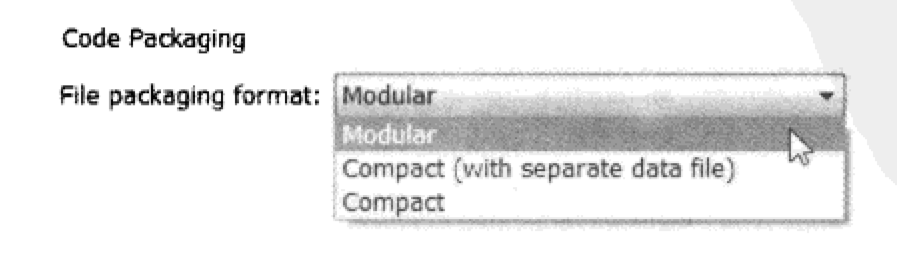


图 5.2.38 选择生成代码的形式

表 5.2.1　不同格式的代码生成情况

File Packaging Format	生成的文件	移除的文件
Modular (default)	model. c model. h model_types. h model_private. h model_data. c (非必需) 子系统文件(非必需)	无
Compact (with separate data file)	model. c model. h model_data. c(非必需)	model_private. h model_types. h
Compact	model. c model. h	model_data. c model_private. h model_types. h

表 5.2.1 列出了代码生成时被移除的文件，但并不表示它们所实现的功能也被移除了，RTW-EC 在生成代码时将这些功能整合进了 model. c 或 model. h，如表 5.2.2 所列。

表 5.2.2　移除与移入文件的关系

移除的文件	移入的文件
model_private. h	model. c 与 model. h
model_types. h	model. h
model_data. c	model. c

5.2.3　脉宽调制

1. 脉宽调制状态图

PWM 技术利用微处理器的数字输出量，有效地使数字电路具备了控制模拟电路的功能，广泛应用于测控领域。本例中，用 8 位二进制控制信号调节 PWM 信号的占空比。

读者参考第 1 章和第 3 章的内容以及 PWM 的原理，可以很快得到图 5.2.39 所示的脉宽调制状态图。

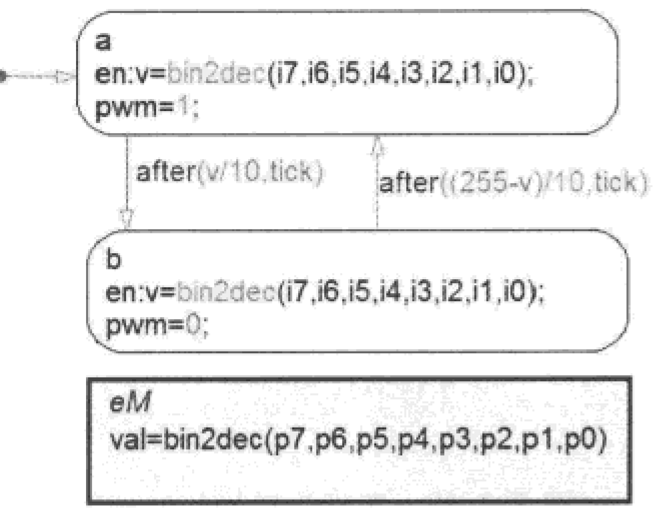


图 5.2.39　脉宽调制状态图



其中的 Embedded MATLAB 函数实现由二进制转换为十进制的功能。代码如下：

```
function val = bin2dec(p7,p6,p5,p4,p3,p2,p1,p0)
val = p7 * 2^7 + p6 * 2^6 + p5 * 2^5 + p4 * 2^4 + p3 * 2^3 + p2 * 2^2 + p1 * 2^1 + p0 * 2^0;
end
```

所用数据如图 5.2.40 所示，其中 i0～i7 表示输入的控制信号，v 为中间变量，用来表示脉冲的占空比，PWM 为输出信号。

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledTyp
i0	Input	1		double			double
i1	Input	2		double			double
i2	Input	3		double			double
i3	Input	4		double			double
i4	Input	5		double			double
i5	Input	6		double			double
i6	Input	7		double			double
i7	Input	8		double			double
pwm	Output	1	☐	double			double
v	Local		☐	double			double

图 5.2.40　状态图数据列表

2. 功能验证模型

完成 Stateflow 状态图之后，在 Simulink 模块库中找到图 5.2.41、图 5.2.42 所示模块，并按图 5.2.43 所示连接。

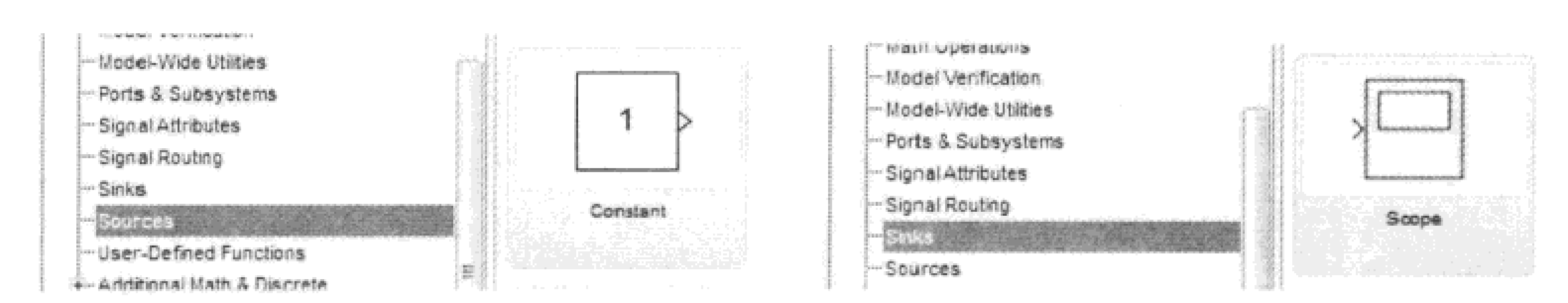


图 5.2.41　Constant 模块　　　　图 5.2.42　Scope 模块

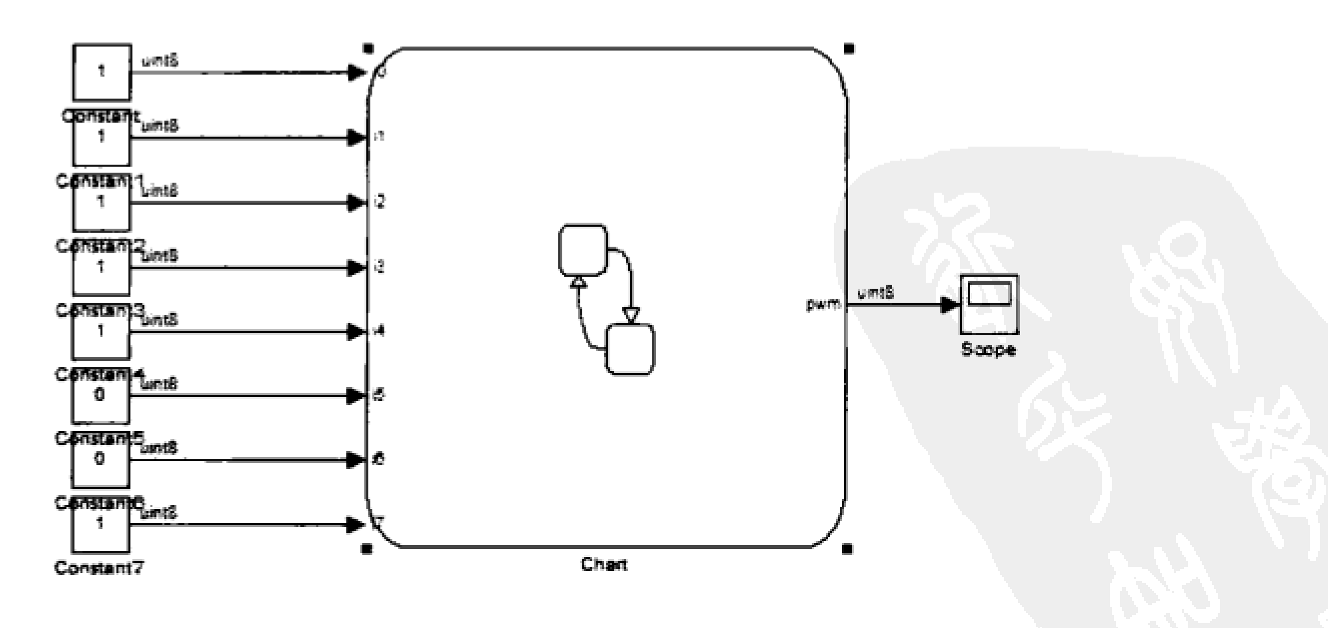
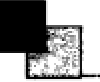


图 5.2.43　功能验证模型



完成以上设置后执行仿真，输入端模拟外部控制信号，在输入端任意设定两组不同的电平值，输出脉冲的占空比如图 5.2.44、图 5.2.45 所示。

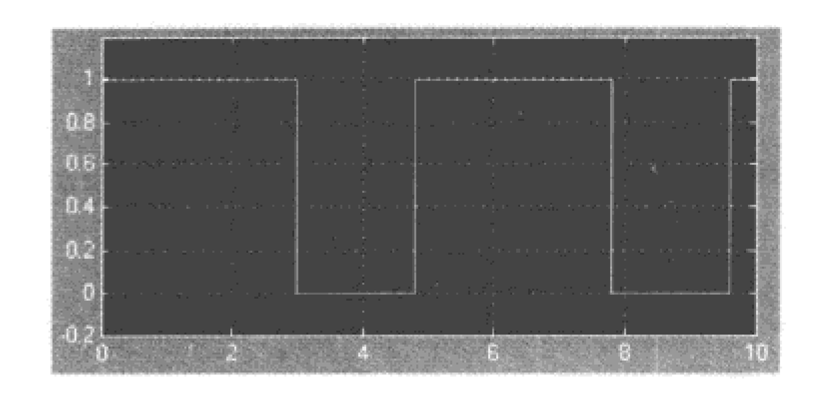



图 5.2.44　仿真结果 1

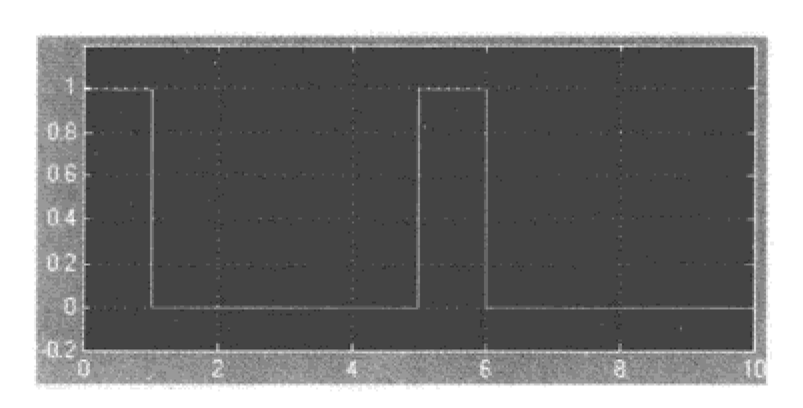


图 5.2.45　仿真结果 2

3. 软件在环测试

软件在环测试(SIL)是在主机上对仿真中生成的函数或手写代码进行非实时性联合仿真评估，当软件组件包含需要在目标平台上执行的生成代码和手写代码的组合时，应该考虑进行软件在环测试，完成对模型生成代码的早期验证。

软件在环测试不需要硬件，只是对算法代码进行测试，具体做法是对要进行测试的子系统编译可生成 SIL 模块，比较原模块与 SIL 模块的输出，以此确认算法的正确性。

在模块库 Simulink/Ports & Subsystems 中找到输入模块与输出模块，替换功能验证模型中的 Constant 和 Scope 模块，如图 5.2.46 所示。

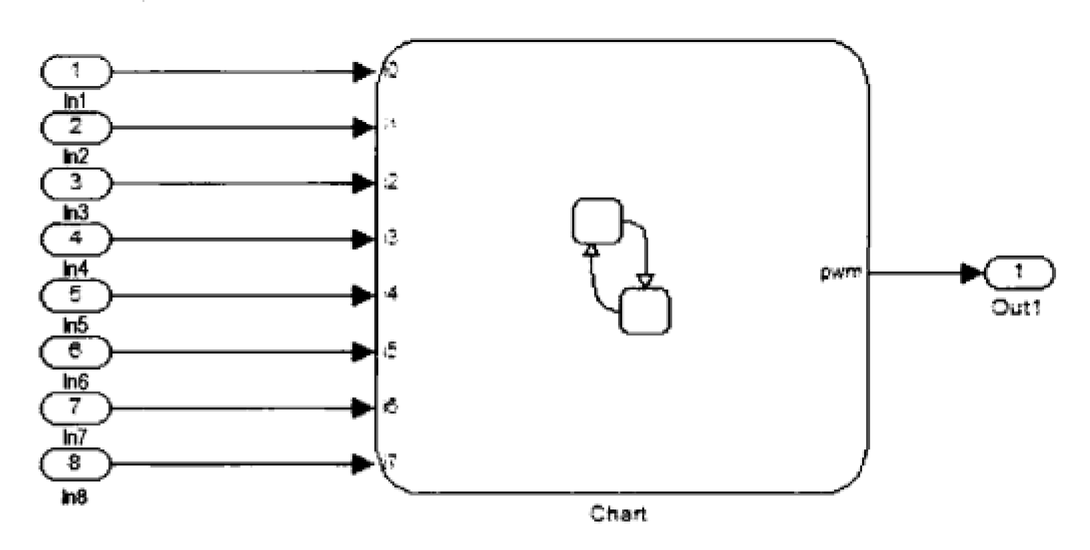


图 5.2.46　代码生成模型

另外，还需要确保模型中数据的类型均为 uint8，这里推荐用户使用模型浏览器批量修改 Simulink 端口与 Stateflow 数据的数据类型，既快捷又可避免遗漏。

单击模型窗口的按钮，打开模型浏览器，将流水灯状态图里各变量的数据类型设置为 uint8(图 5.2.47)。

将流水灯状态图里，Embedded MATLAB 函数模块内的变量数据类型设置为 uint8(图 5.2.48)。

打开 Embedded MATLAB 函数，将其中所有常数的类型也转换为 uint8：

```
function val = bin2dec(p7,p6,p5,p4,p3,p2,p1,p0)
val = p7 * uint8(2^7) + p6 * uint8(2^6) + p5 * uint8(2^5) + p4 * uint8(2^4) + p3 * uint8(2^3) + p2 *
uint8(2^2) + p1 * uint8(2^1) + p0 * uint8(2^0);
end
```



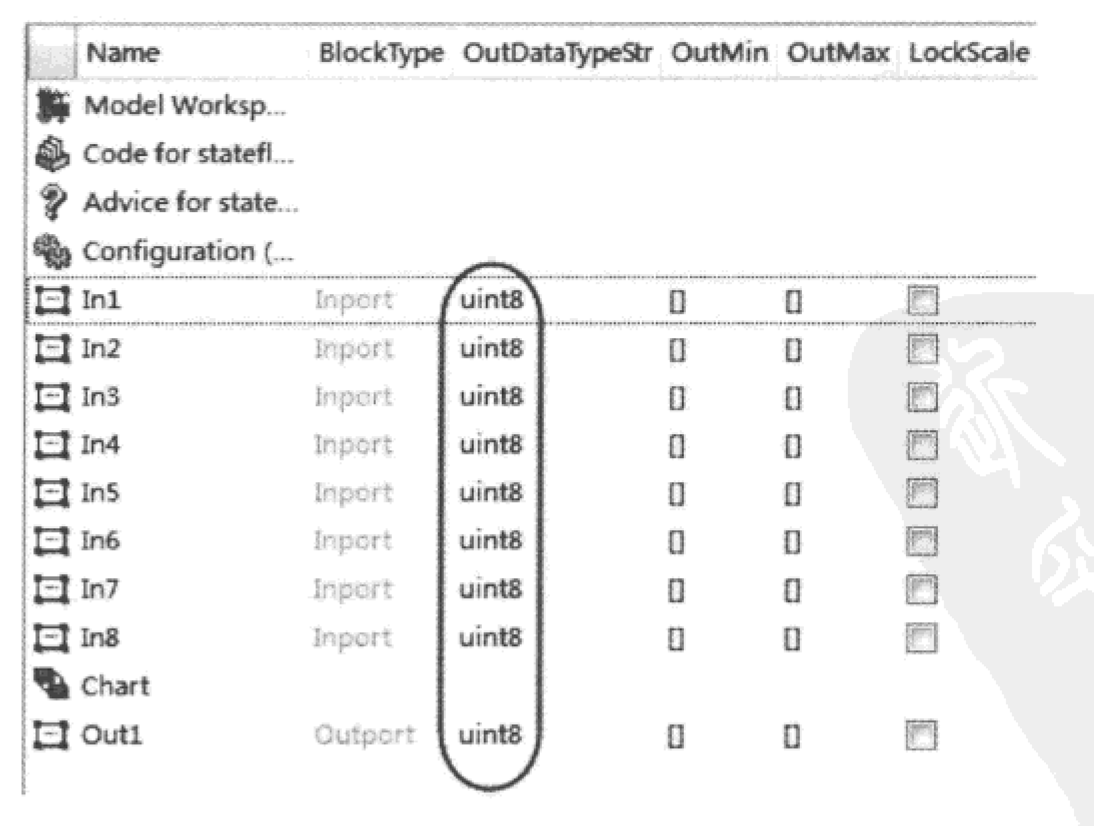
Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType
pwm	Out...	1	☐	uint8			unknown
i0	Input	1		uint8			unknown
i1	Input	2		uint8			unknown
i2	Input	3		uint8			unknown
i3	Input	4		uint8			unknown
i4	Input	5		uint8			unknown
i5	Input	6		uint8			unknown
i6	Input	7		uint8			unknown
i7	Input	8		uint8			unknown
v	Local		☐	uint8			unknown

图 5.2.47　设置模型状态图数据类型

Name	Scope	DataType	CompiledType	Port
p0	Input	uint8	unknown	
p1	Input	uint8	unknown	
p2	Input	uint8	unknown	
p3	Input	uint8	unknown	
p4	Input	uint8	unknown	
p5	Input	uint8	unknown	
p6	Input	uint8	unknown	
p7	Input	uint8	unknown	
val	Output	uint8	unknown	

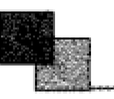
图 5.2.48　设置 Embedded MATLAB 函数变量的数据类型

代码生成模型中的 In 模块的数据类型同样设置为 uint8，Out 模块的数据类型可设为自动继承，也可强制设置为 uint8(图 5.2.49)。



Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockScale
Model Worksp...					
Code for statefl...					
Advice for state...					
Configuration (...					
In1	Inport	uint8	[]	[]	☐
In2	Inport	uint8	[]	[]	☐
In3	Inport	uint8	[]	[]	☐
In4	Inport	uint8	[]	[]	☐
In5	Inport	uint8	[]	[]	☐
In6	Inport	uint8	[]	[]	☐
In7	Inport	uint8	[]	[]	☐
In8	Inport	uint8	[]	[]	☐
Chart					
Out1	Outport	uint8	[]	[]	☐

图 5.2.49　设置模型端口数据类型



在菜单栏上选择 Simulation→Configuration Parameters，打开模型的参数设置对话框，作以下设置：在 Real-Time Workshop 选项卡，设置 TCL 文件为 ert. tlc，如图 5. 2. 50 所示。

Target selection
System target file: ert.tlc Browse...
Language: C
Description: Real-Time Workshop Embedded Coder

图 5. 2. 50 选择 TLC 文件

在 Solver 面板，设置求解器为定步长离散求解器，如图 5. 2. 51 所示。

Solver options
Type: Fixed-step Solver: discrete (no continuous states)
Fixed-step size (fundamental sample time): auto

图 5. 2. 51 求解器设置

在 Report 面板，勾选所有复选框，便于后期检查及跟踪，如图 5. 2. 52 所示。

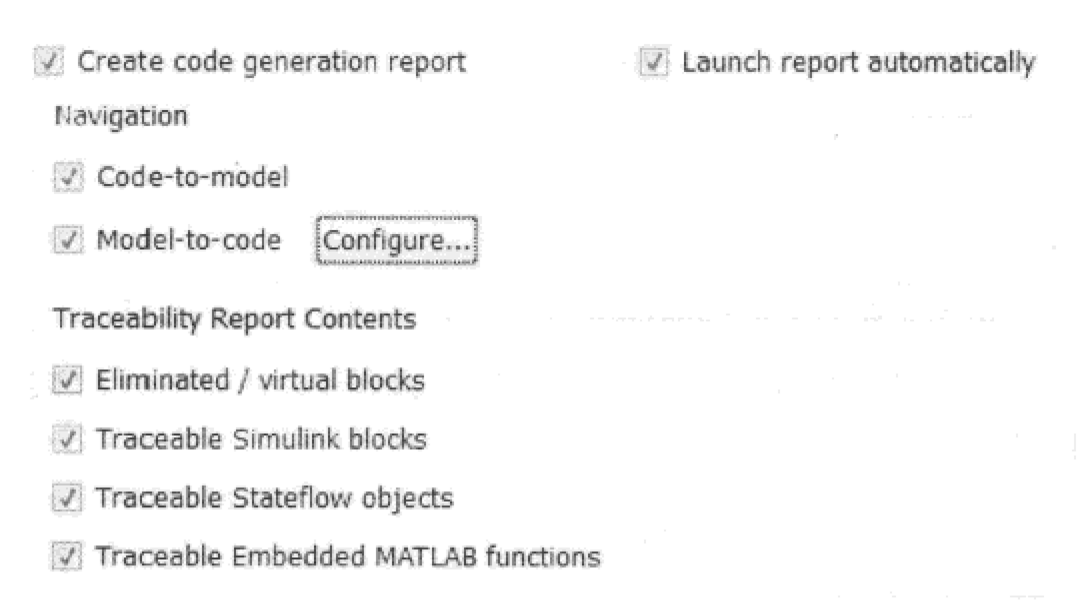


图 5. 2. 52 生成报告设置

在 Real-Time Workshop→SIL and PIL Verification 面板的 Create block 下拉列表中，选择 SIL 选项，如图 5. 2. 53 所示。

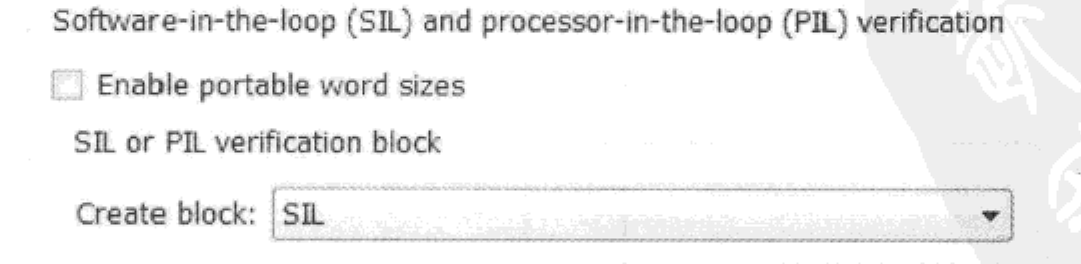
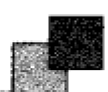


图 5. 2. 53 选择 SIL 选项

单击模型工具栏的按钮，生成 SIL 模块，如图 5. 2. 54 所示。



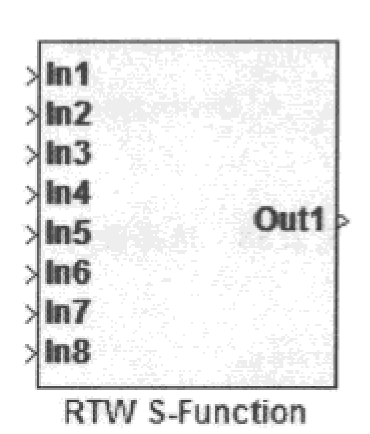


图 5.2.54　软件在环测试模块

以 SIL 模块替换图 5.2.43 中的 Stateflow 模块，重建验证模型，如图 5.2.55 所示。

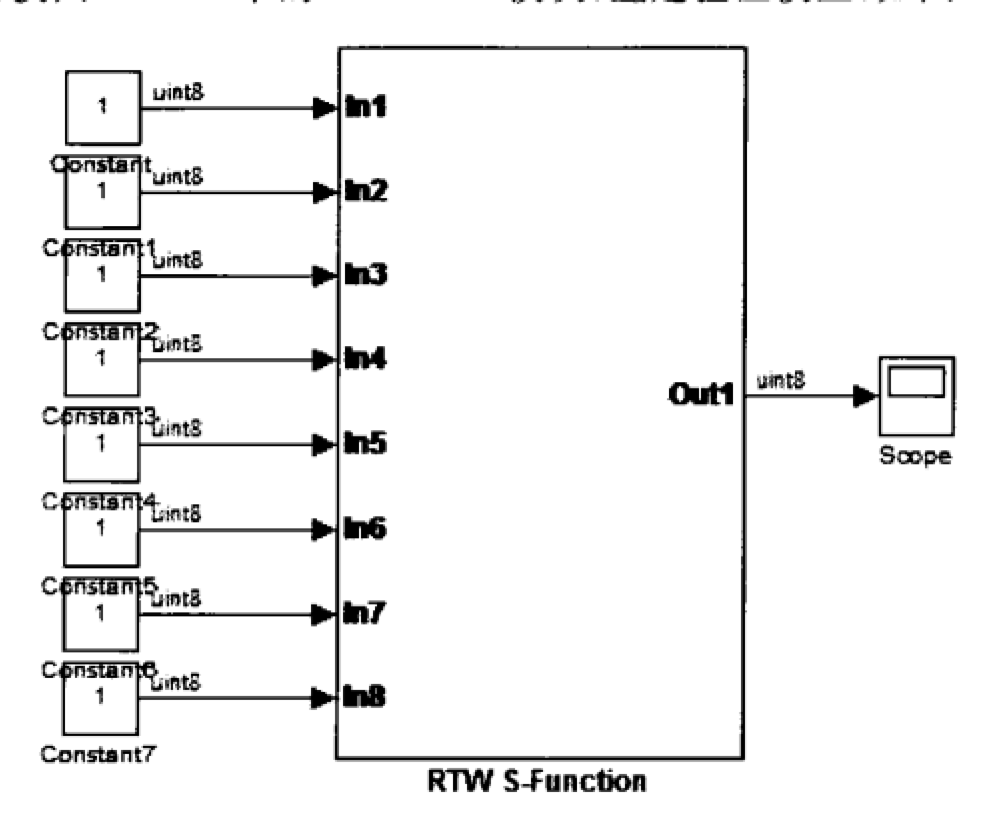


图 5.2.55　软件在环测试模型

再次输入功能验证模型中的两组电平，可得到相同的脉冲输出。说明 SIL 模块实现了 Stateflow 模块的功能，如图 5.2.56、图 5.2.57 所示。

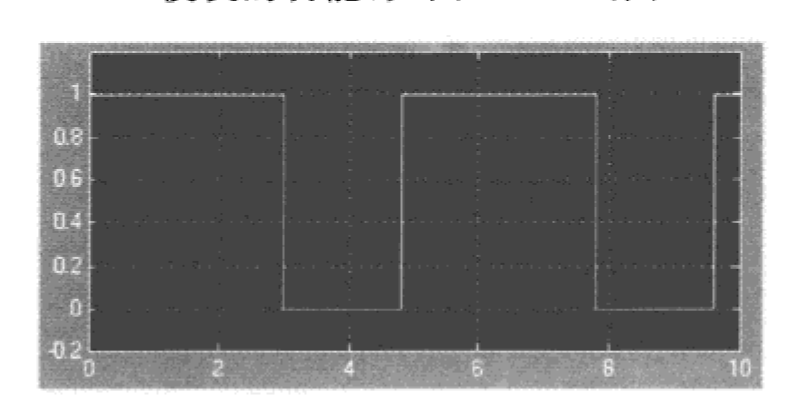


图 5.2.56　软件在环测试结果 1

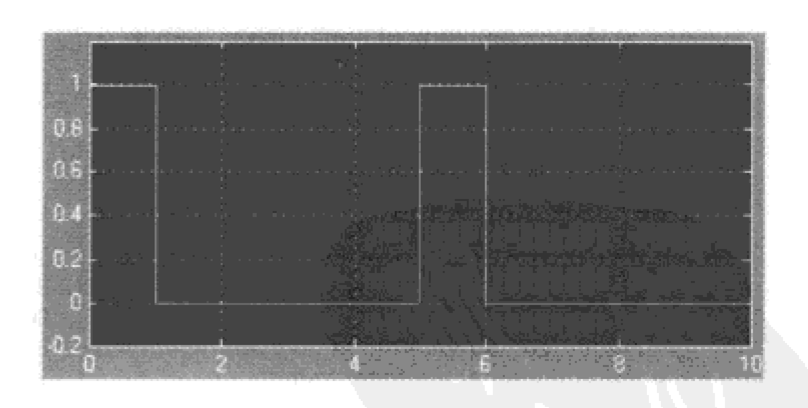
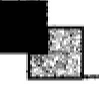


图 5.2.57　软件在环测试结果 2

4. 自动生成代码

打开模型的参数设置对话框，在 Hardware Implimentation 面板中，设置器件类型为 8051，如图 5.2.58 所示。

单击模型工具栏的按钮，生成代码的报告如图 5.2.59 所示。



Embedded hardware (simulation and code generation)

Device vendor: Intel Device type: 8051 Compatible

图 5.2.58 选择硬件类型



图 5.2.59 代码报告

打开 Keil uVision4，选择芯片 AT89C51，建立工程，并加入生成的代码，如图 5.2.60 所示。

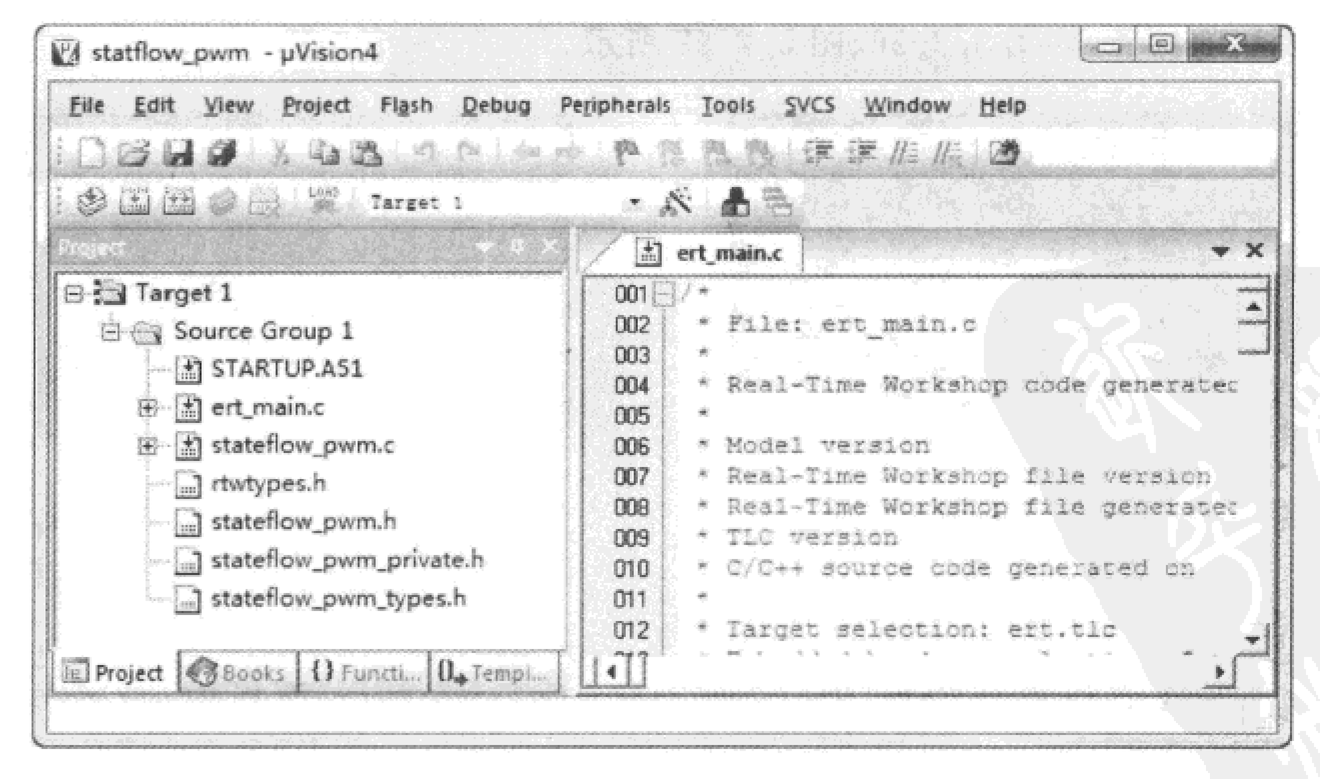
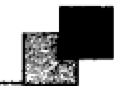


图 5.2.60 建立 Keil 工程



为了使自动生成的代码与实际硬件的端口、寄存器等能有效对应，需对 ert_main.c 文件作一些修改，如下所示：

```
...
// #include <stdio.h>                    // 删除该头文件
#include "stateflow_pwm.h"
#include "rtwtypes.h"
#include <REGX51.H>                      // 添加 51 头文件
...
    /* Set model inputs here */
    stateflow_pwm_U.In1 = P1_0;          // 将模型输入与硬件接口相关联
    stateflow_pwm_U.In2 = P1_1;
    stateflow_pwm_U.In3 = P1_2;
    stateflow_pwm_U.In4 = P1_3;
    stateflow_pwm_U.In5 = P1_4;
    stateflow_pwm_U.In6 = P1_5;
    stateflow_pwm_U.In7 = P1_6;
    stateflow_pwm_U.In8 = P1_7;
/* Step the model */
 stateflow_pwm_step();
/* Get model outputs here */
 P3_0 = stateflow_pwm_Y.Out1;            // 将模型输出与硬件接口相关联
...
int_T main();                            // 删除主函数的参数
int_T main()                             // 删除主函数的参数
{  ...
   // 删除以下代码
   //printf("Warning: The simulation will run forever. "
   //"Generated ERT main won't simulate model step behavior. "
   //"To change this behavior select the 'MAT-file
   //logging' option.\n");
   //fflush((NULL));
   while (rtmGetErrorStatus(stateflow_pwm_M) == (NULL))
   {  /* Perform other application tasks here */
   rt_OneStep();                         //调用 rt_OneStep 函数
   }
...
}
```

单击 Keil 工具栏的按钮，或按 Alt+F7 组合键，在 Output 选项卡，选择 Create HEX File 选项，并指定 HEX 文件的格式 HEX-80，如图 5.2.61 所示。

再单击工具栏按钮，重编译工程，窗口下部的信息显示已成功生成.hex 文件，如图5.2.62所示。

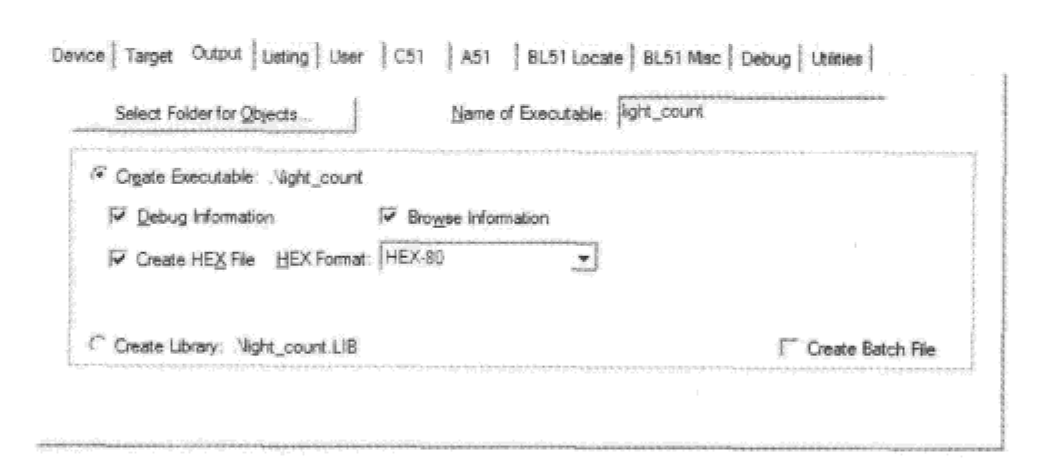


图 5.2.61　设置输出格式

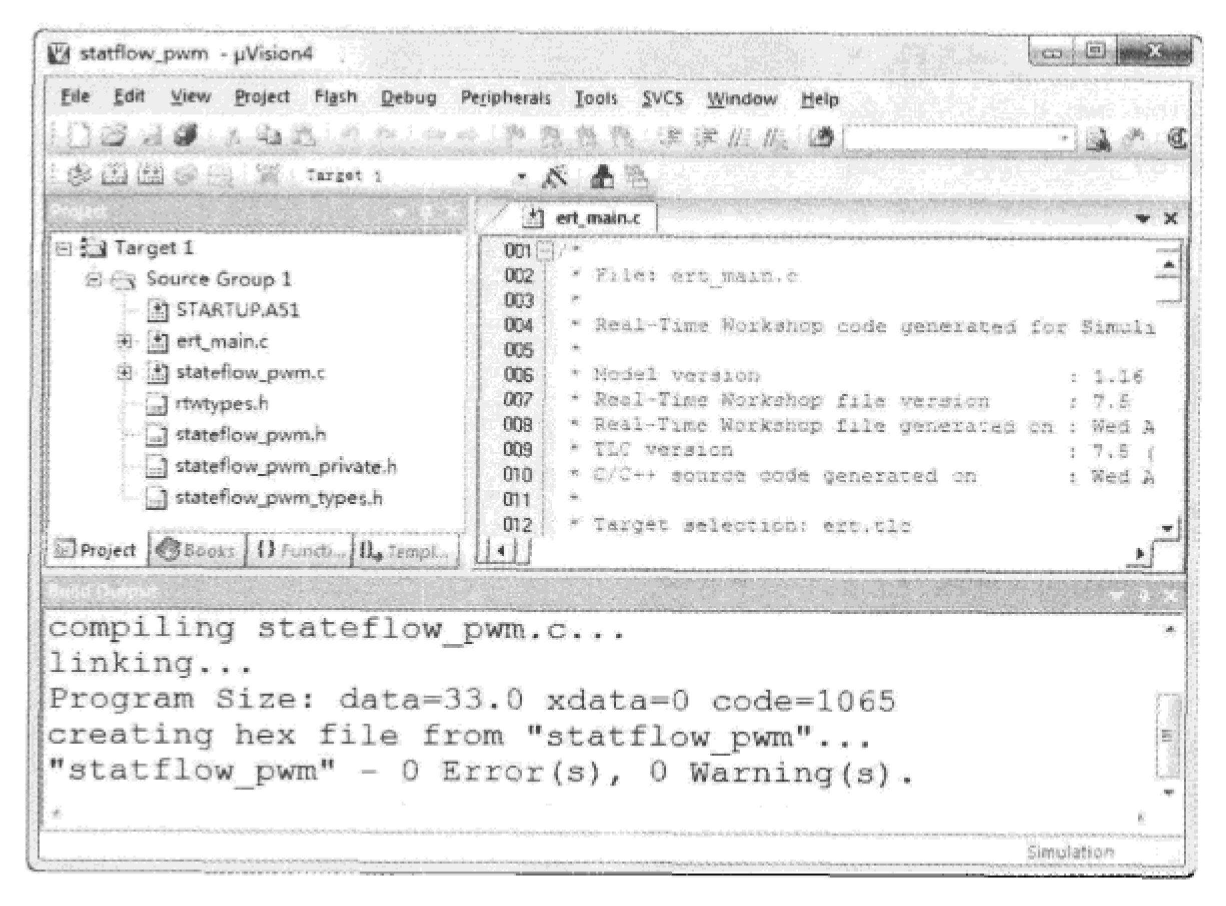


图 5.2.62　编译生成 HEX 文件

5. 虚拟硬件测试

根据本章第 5.1 节的介绍，建立 Proteus 模型，并在 P3_0 口接入虚拟仪器 Oscilloscope 来观察输出的 PWM 波形，如图 5.2.63 所示。

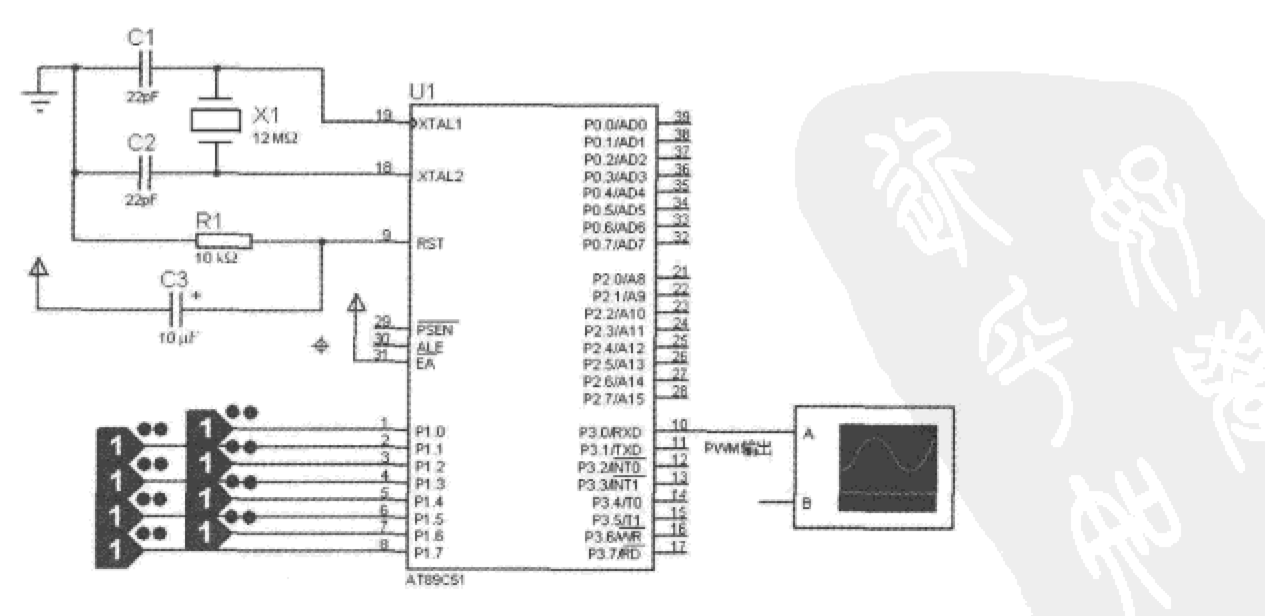
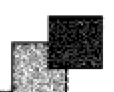


图 5.2.63　脉宽调制 Proteus 模型



加载 HEX 文件并执行仿真后，通过调节控制电平，可以从虚拟示波器中看到脉冲占空比的变化。如图 5.2.64、图 5.2.65 所示。

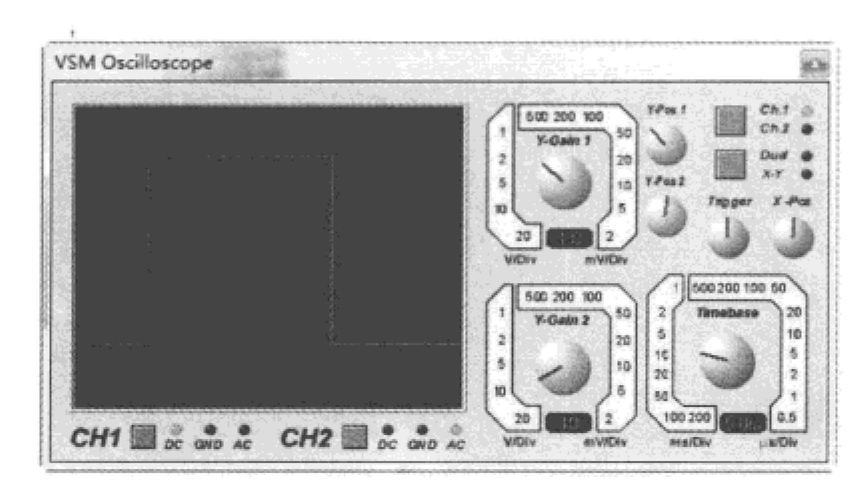


图 5.2.64　虚拟硬件测试结果 1

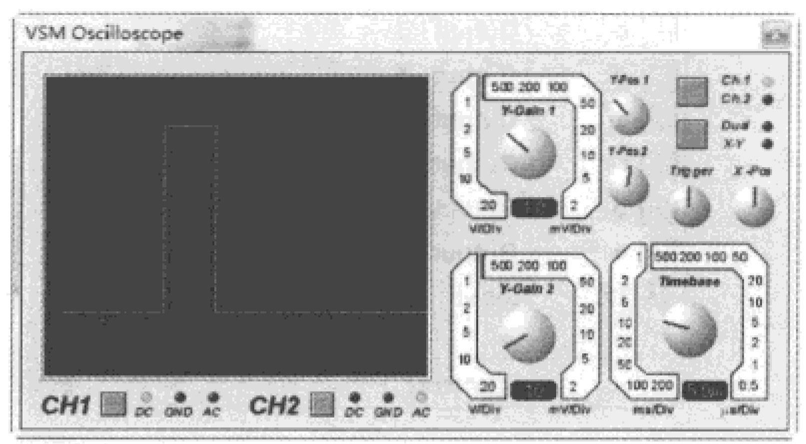


图 5.2.65　虚拟硬件测试结果 2

通过上述一系列测试，证明了基于模型的 C51 单片机的代码快速生成是可行的。手工代码加自动模型代码的方式，是近几年才开始流行的开发嵌入式系统的新技术，可大大提高开发效率，实现从概念到实现的设计理念。

5.2.4　流水灯

1. 流水灯状态图

流水灯是单片机的常见应用，其特点是若干个灯泡按设定的时间依次点亮。根据第 3 章的介绍，很容易建立流水灯的状态图，如图 5.2.66 所示。

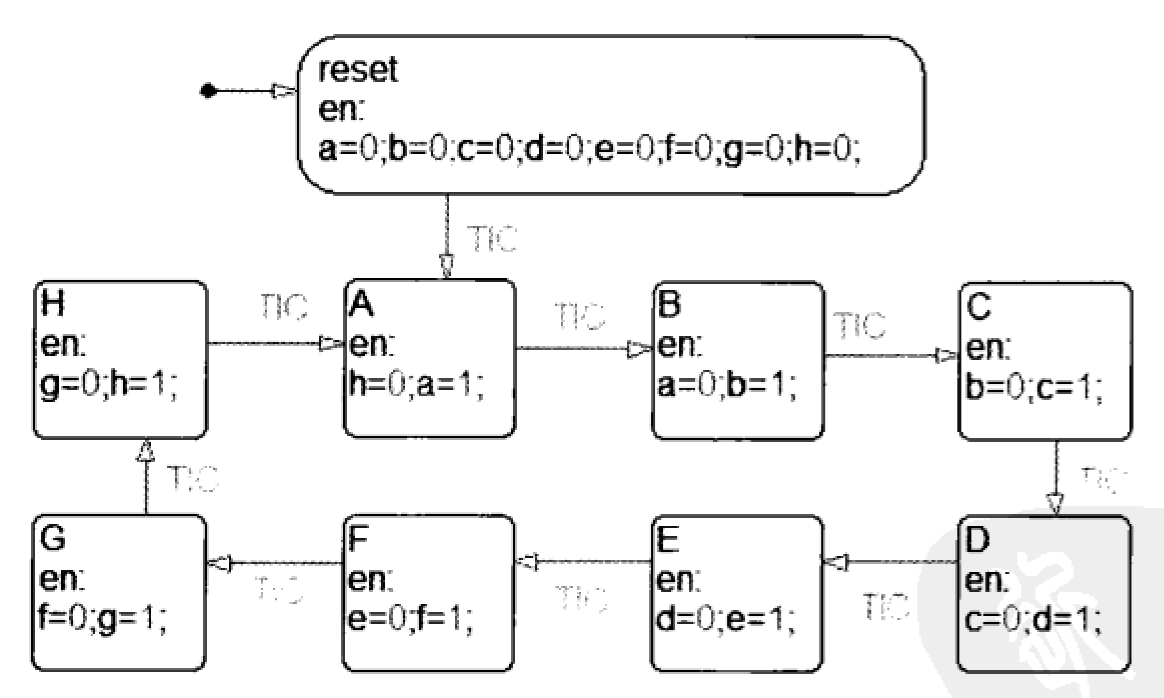


图 5.2.66　流水灯状态图

其中输出变量 a、b...g、h 表示 8 个灯泡的点亮状态，事件 TIC(上升沿)表示计数时间到，如图 5.2.67 所示。

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	CompiledSize	Trigger
a	Output	1	☐	double			uint8	1	
b	Output	2	☐	double			uint8	1	
c	Output	3	☐	double			uint8	1	
d	Output	4	☐	double			uint8	1	
e	Output	5	☐	double			uint8	1	
f	Output	6	☐	double			uint8	1	
g	Output	7	☐	double			uint8	1	
h	Output	8	☐	double			uint8	1	
TIC	Input	1							Rising

Contents　Search Results

图 5.2.67　状态图数据与事件列表

2. 功能验证模型

完成 Stateflow 状态图之后，在 Simulink 模块库中找到图 5.2.68～图 5.2.70 所示模块，并按图 5.2.71 连接。

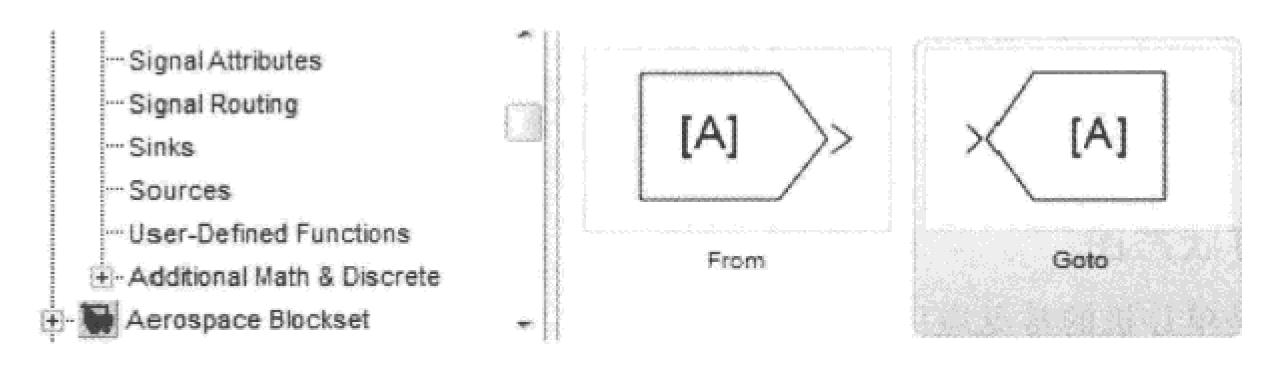


图 5.2.68　From 与 Goto 模块

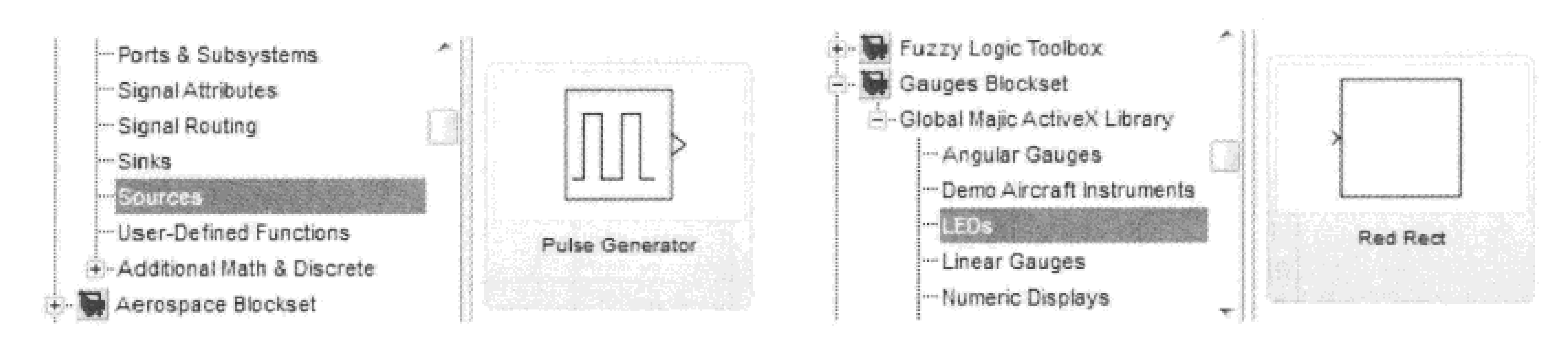


图 5.2.69　脉冲发生器模块　　　　图 5.2.70　红色 LED 模块

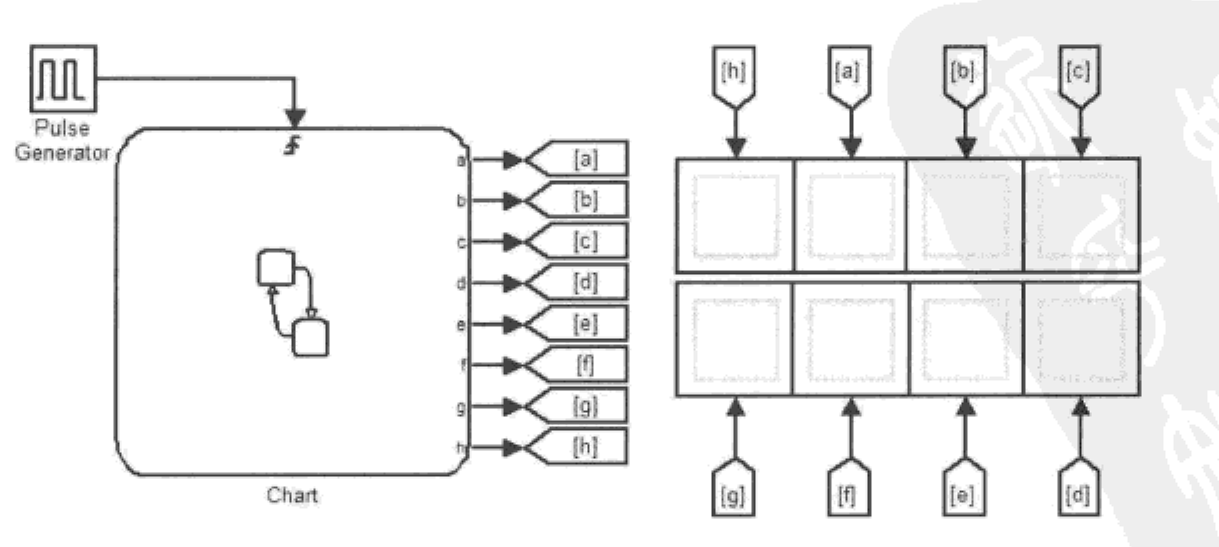
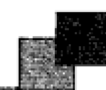


图 5.2.71　功能验证模型



选择模型主窗口的菜单项 Simulation→Configuration Parameters…，打开模型参数对话框，在 Solver options 面板中，设置求解器为定步长离散求解器，步长为 0.01，如图 5.2.72 所示。

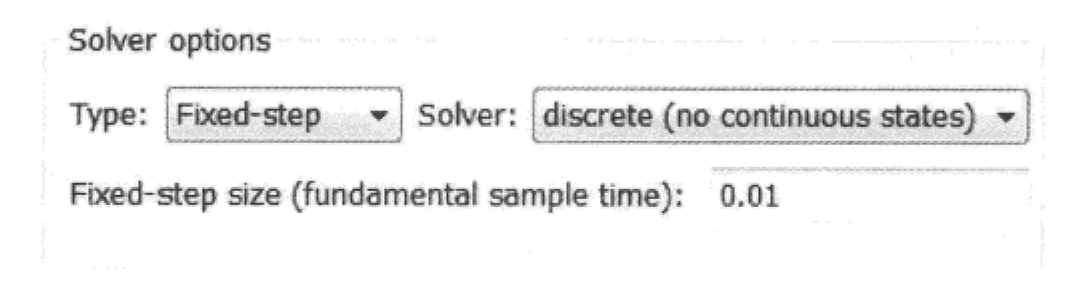


图 5.2.72　求解器设置

双击脉冲发生器模块，根据实际需要设置脉冲周期与脉冲宽度(图 5.2.73)。

Parameters

Pulse type: Time based

Time (t): Use simulation time

Amplitude:

1

Period (secs):

5

Pulse Width (% of period):

10

Phase delay (secs):

0

☑ Interpret vector parameters as 1-D

图 5.2.73　设置脉冲周期与脉冲宽度

完成以上设置后执行仿真，模型中的 LED 模块顺序点亮，如图 5.2.74 所示。

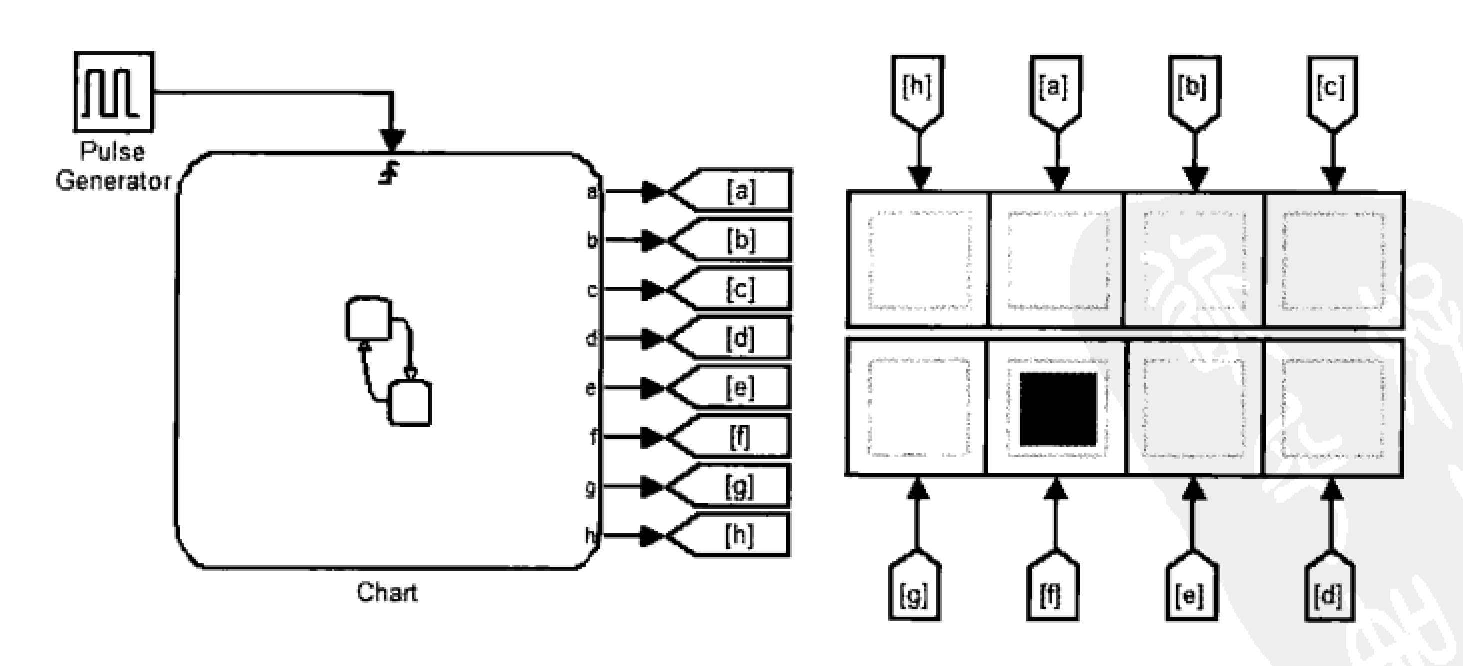


图 5.2.74　功能验证仿真结果

3. 软件在环测试

(1) 数据类型转换。在模块库 Simulink→Ports & Subsystems 中找到输入模块与输出模块，替换图 5.2.71 中的脉冲生成器与 Goto 模块，删去 From 与 LED 模块，并将模型另存。

单击模型窗口的按钮，打开模型浏览器，将流水灯状态图里各变量的数据类型设置为 uint8(图 5.2.75)。

功能验证模型中的 In 模块的数据类型同样设置为 uint8，Out 模块的数据类型可设为自动继承，也可强制设置为 uint8，如图 5.2.76 所示。

Name	Scope	Port	Resolve Signal	DataType	Trigger
TIC	Input	1			Rising
a	Output	1	☐	uint8	
b	Output	2	☐	uint8	
c	Output	3	☐	uint8	
d	Output	4	☐	uint8	
e	Output	5	☐	uint8	
f	Output	6	☐	uint8	
g	Output	7	☐	uint8	
h	Output	8	☐	uint8	

图 5.2.75　设置模型状态图中的数据类型

Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockS
Model...					
Code f...					
Advice...					
Config...					
In1	Inport	uint8	[]	[]	☐
Chart					
Out1	Outport	Inherit: auto	[]	[]	☐
Out2	Outport	Inherit: auto	[]	[]	☐
Out3	Outport	Inherit: auto	[]	[]	☐
Out4	Outport	Inherit: auto	[]	[]	☐
Out5	Outport	Inherit: auto	[]	[]	☐
Out6	Outport	Inherit: auto	[]	[]	☐
Out7	Outport	Inherit: auto	[]	[]	☐
Out8	Outport	Inherit: auto	[]	[]	☐

图 5.2.76　设置模型端口的数据类型

修改后的模型如图 5.2.77 所示。

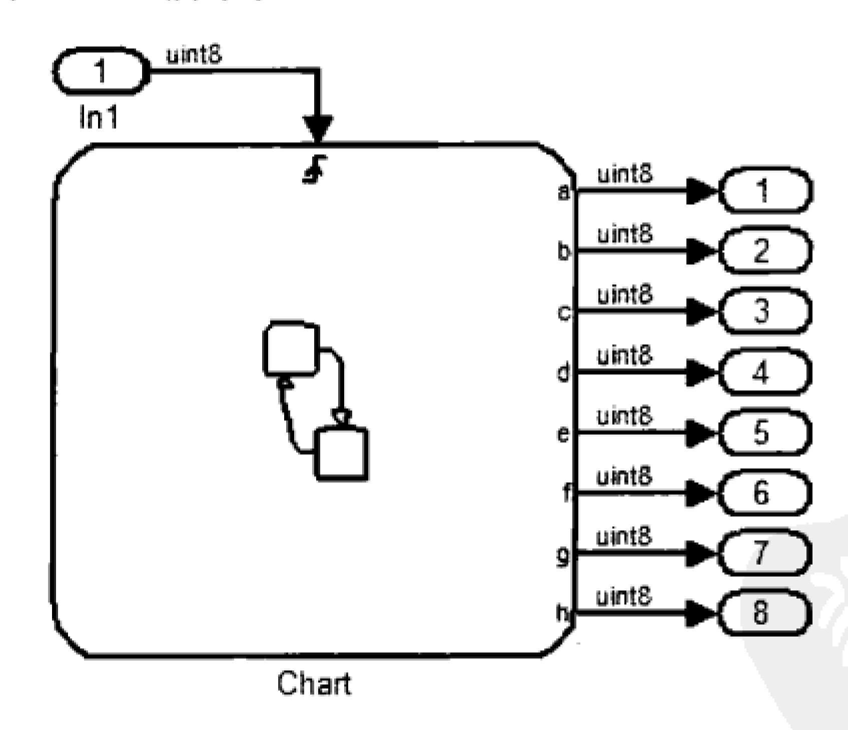
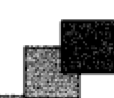


图 5.2.77　代码生成模型

(2) 模型参数设置。打开模型参数对话框，在 Real-Time Workshop 面板中设置 TLC 文件为 ert.tlc，如图 5.2.78 所示。

Real-Time Workshop→Interface 面板，取消勾选不必要的复选框，如图 5.2.79 所示。

Real-Time Workshop→Report 面板，勾选所有复选框，便于后期检查及跟踪，如图 5.2.80 所示。



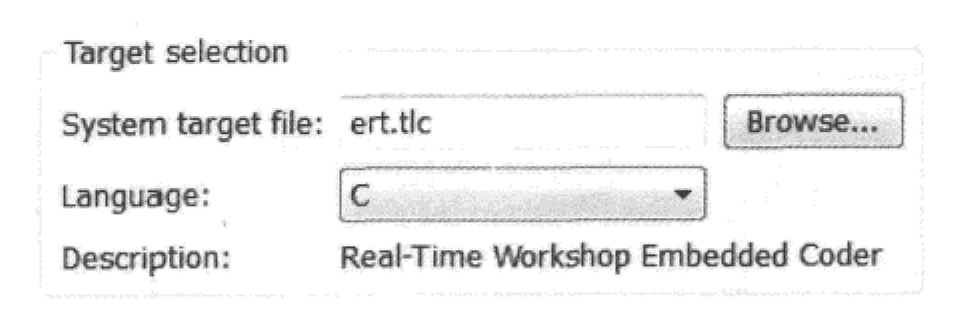


图 5.2.78　选择 TLC 文件

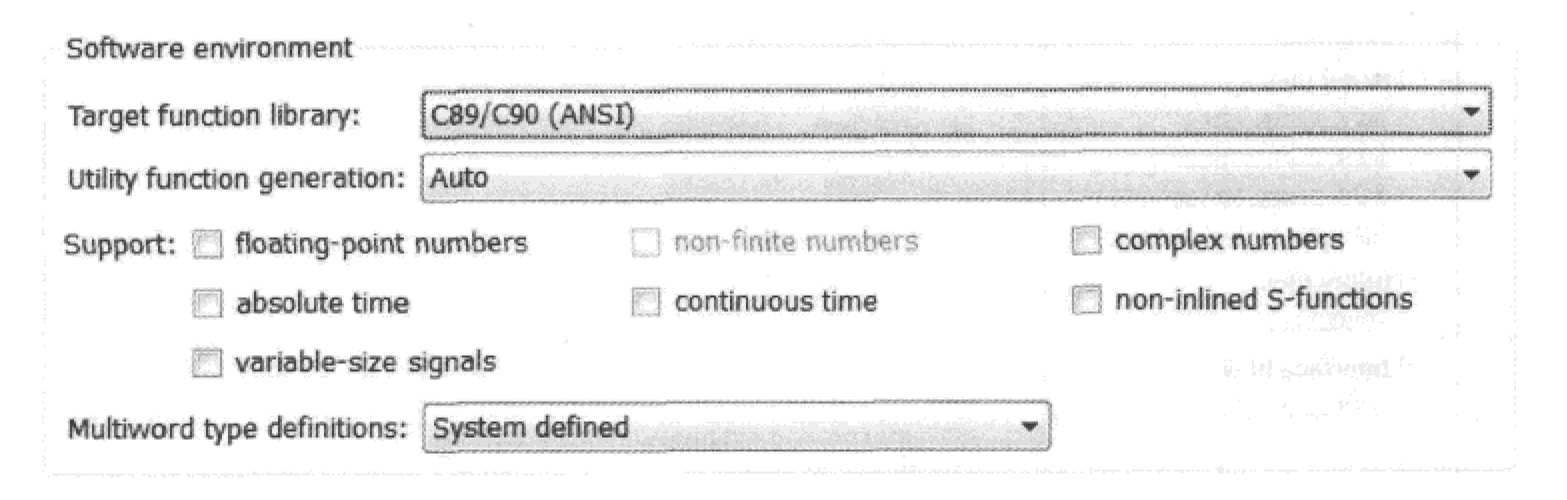


图 5.2.79　Interface 面板

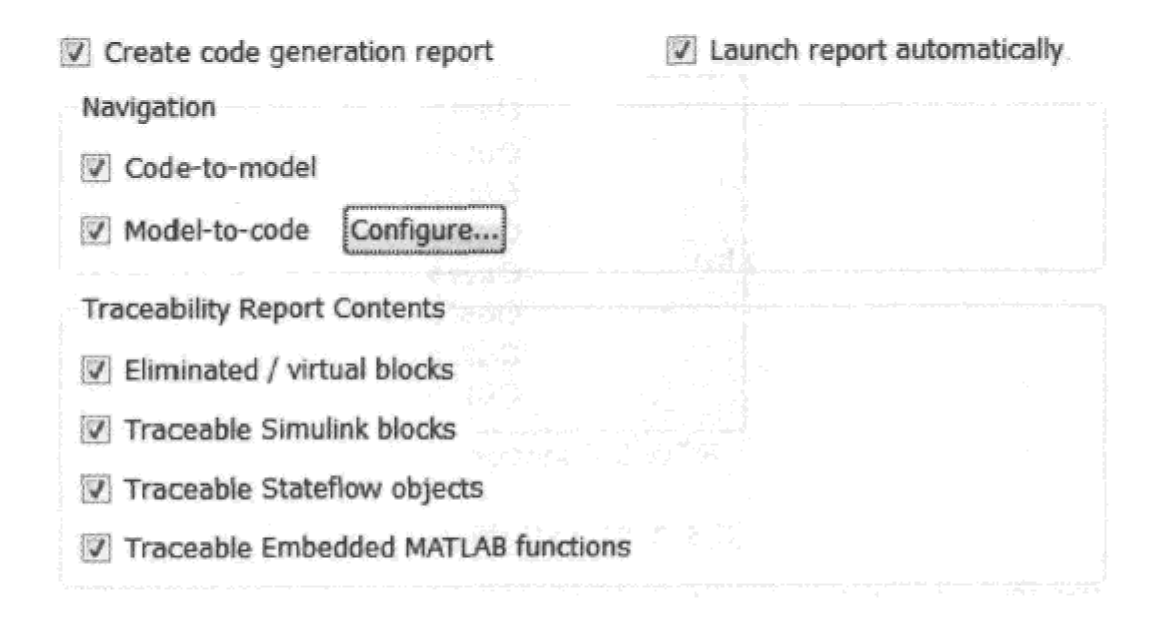


图 5.2.80　生成报告设置

(3) 生成 SIL 模块。在 Real-Time Workshop→SIL and PIL Verification 面板的 Create block 选项，选择 SIL 选项，如图 5.2.81 所示。

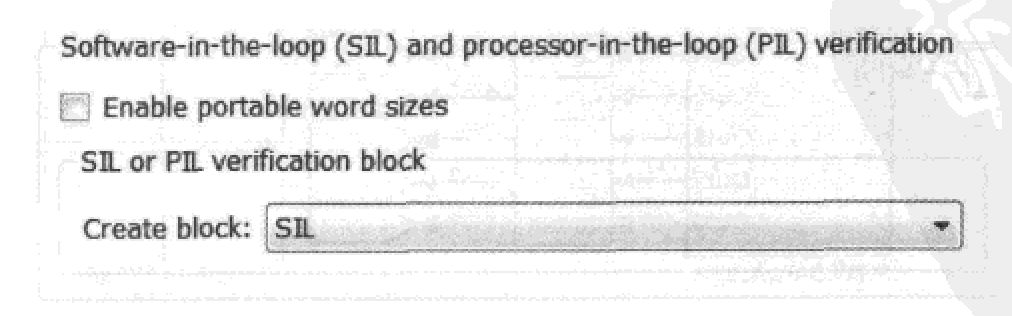


图 5.2.81　选择 SIL 选项

之后单击模型工具栏的按钮，得到代码生成报告与 SIL 模块，如图 5.2.82 和图 5.2.83 所示。

Back | Forward

Contents
Summary
Subsystem Report
Code Interface Report
Traceability Report

Generated Files
[-] **Main file**
ert_main.c
[-] **Model files**
light_count_keil.c
light_count_keil.h
light_count_keil_private.h
light_count_keil_types.h
[-] **Utility files**
rtwtypes.h
[-] **Interface files**
light_count_keil_sf.c

```
20    */
21
22  #if !defined(S_FUNCTION_NAME)
23  #define S_FUNCTION_NAME              light_count_keil_sf
24  #endif
25
26  #define S_FUNCTION_LEVEL             2
27
28  /* So mdlEnable and mdlDisable will be defined in the simstr
29  #if !defined(RTW_GENERATED_S_FUNCTION)
30  #define RTW_GENERATED_S_FUNCTION
31  #endif
32
33  #include <stdio.h>
34  #include <math.h>
35  #include "simstruc.h"
36  #include "fixedpoint.h"
37  #define rt_logging_h
38  #define RTWSfcnInfo                  void *
39  #include "light_count_keil.h"
40
41  static int8_T light_count_keil_sf_counter = 0;
42
43  /*
44   * This function registers the input and output signal prope
45   * generated ERT code.
46   */
47  static void mdlInitializeSizes(SimStruct *S)
48  {
```

图 5.2.82　代码报告

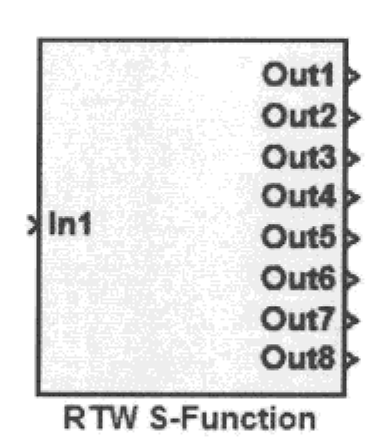


图 5.2.83　SIL 模块

以 SIL 模块替换图 5.2.71 的 Stateflow 模块，重建验证模型，如图 5.2.84 所示，并在各端口间加入必要的数据类型转换模块。该模型的运行结果与图 5.2.74 是一致的。

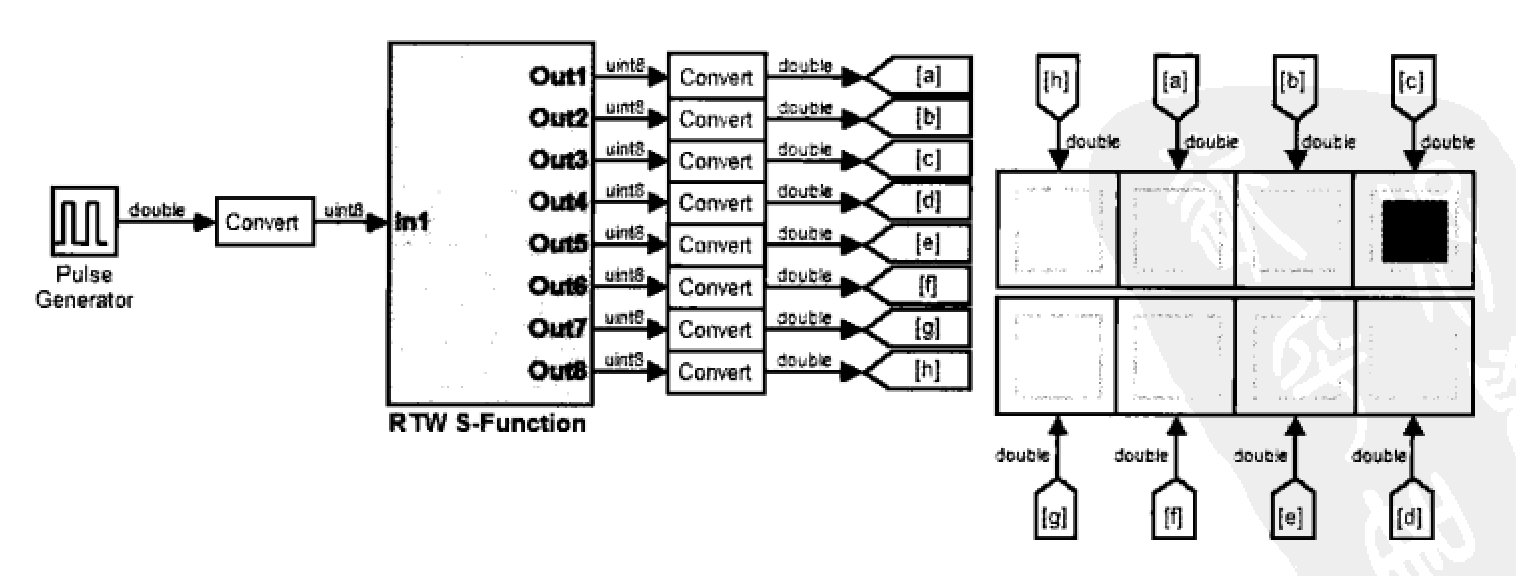
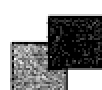


图 5.2.84　软件在环测试结果



4. 自动代码生成及编译

(1) 指定硬件。打开模型参数对话框,在 Hardware Implimentation 面板,设置器件类型为 Intel 8051 Compatible,如图 5.2.85 所示。

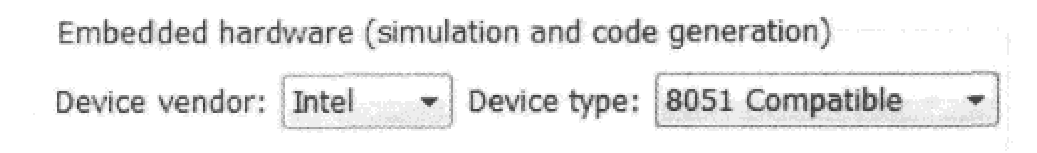


图 5.2.85 选择硬件类型

将 Real-Time Workshop→SIL and PIL Verification 面板的 Create block 选项,恢复成 none,如图 5.2.86 所示。

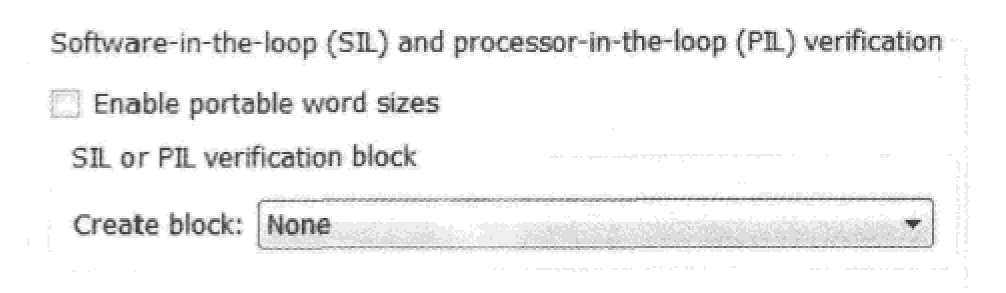


图 5.2.86 取消 SIL 选项

单击模型工具栏的按钮,生成代码,报告如图 5.2.87 所示。

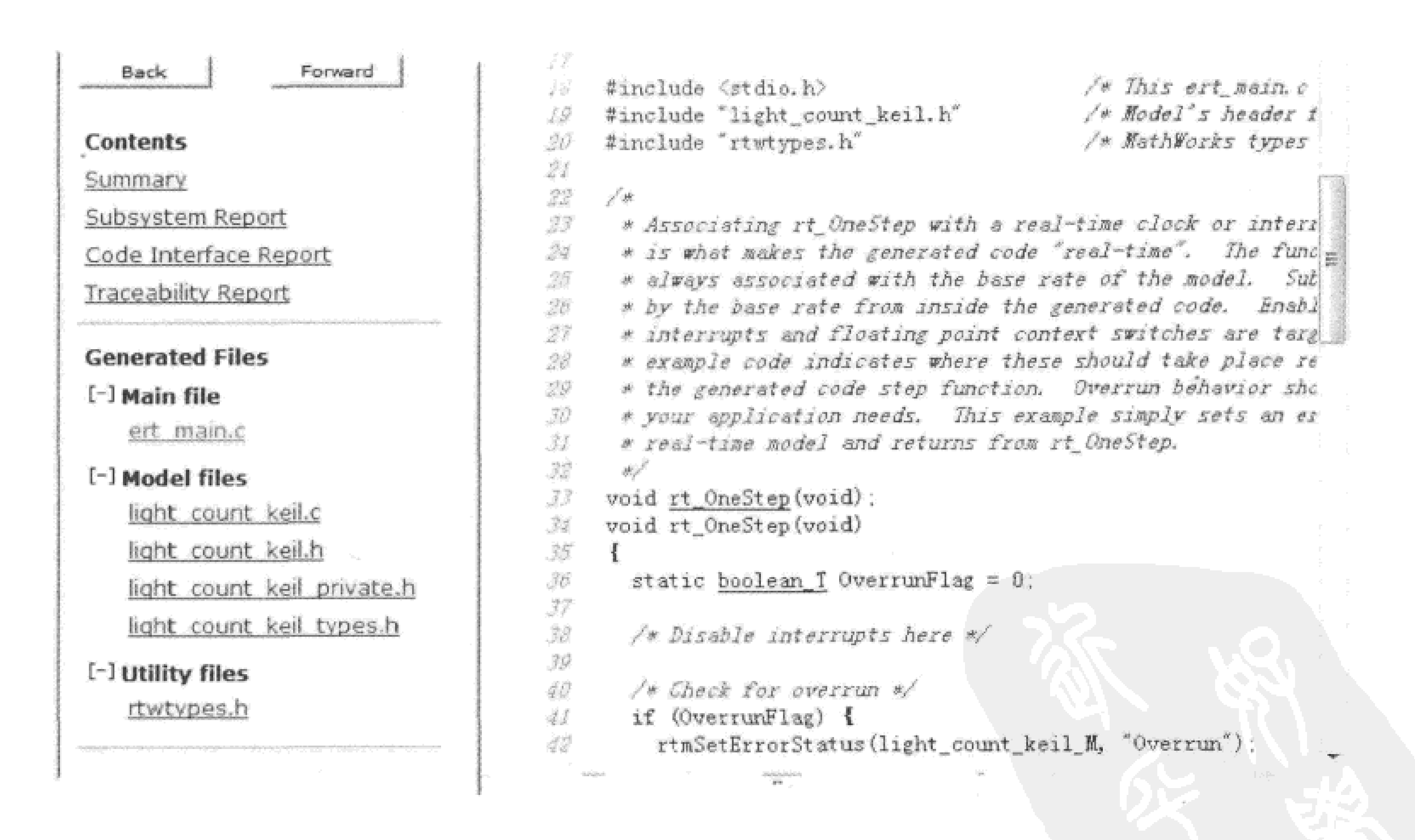


图 5.2.87 代码报告

(2) 建立 Keil 工程。打开 Keil uVision4,建立基于 AT89C51 的工程,将图 5.2.87 所示的 6 个文件与 Keil 工程保存在同一目录下,这样在编译时,编译器会自动找到所需要的头文件,无须手工添加,如图 5.2.88 所示。

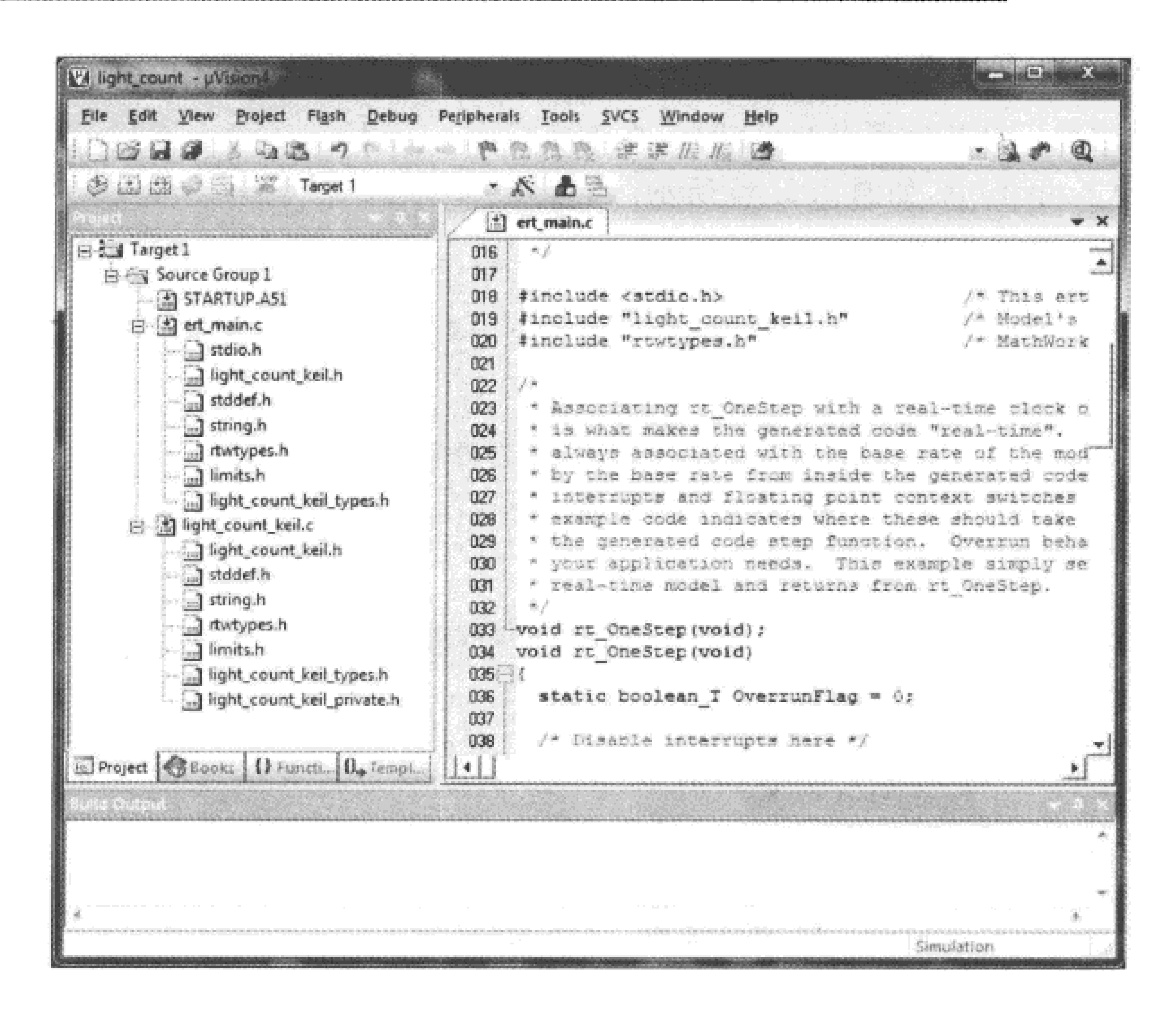


图 5.2.88　建立 Keil 工程

单击 Keil 工具栏的按钮，或按 Alt＋F7 组合键，在 Output 选项卡，勾选 Create HEX File 复选框，并指定 HEX 文件的格式 HEX-80，如图 5.2.89 所示。

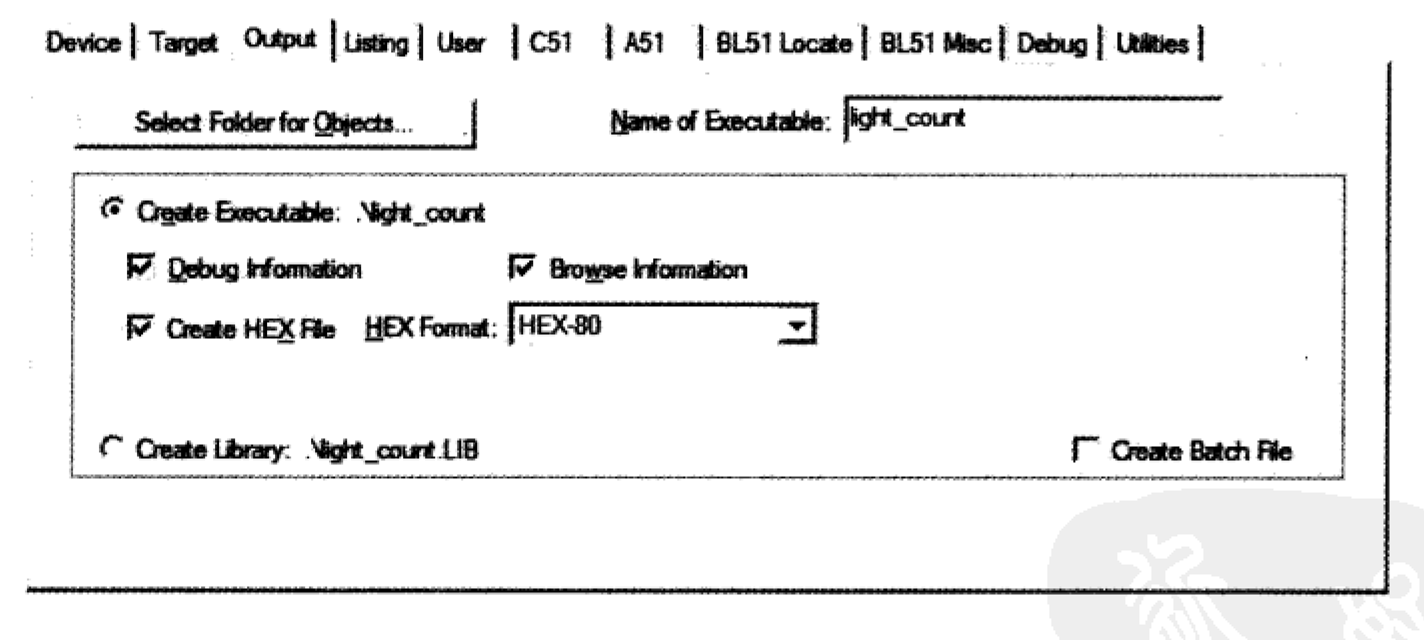


图 5.2.89　设置输出格式

(3) 代码修改。刚刚生成的代码仅实现了 Stateflow 算法，并未包含任何硬件接口，因此还需要将算法的输入/输出与硬件端口相连接，修改的代码以斜体字表示如下：

```
...
#include <stdio.h>                        //删除该头文件
#include "light_count_keil.h"
#include "rtwtypes.h"
```

```
#include <REGX51.H>                    // 新增该头文件
int num = 0;                           // 定时标志
void tic() interrupt 1                 // 定时器 0 中断程序
{
    num++;                             // 定时标志
    TH0 = (65535 - 50000)/256;         // 定时 0.05s 的高位初值
    TL0 = (65535 - 50000)%256;         // 定时 0.05s 的低位初值
}
```

```
void rt_OneStep(void);
void rt_OneStep(void)
{   ...
    // Step the model
    light_count_keil_step();
    // Get model outputs here
    P2_0 = light_count_keil_Y.Out1;
    ...
    P2_7 = light_count_keil_Y.Out8;
    //将输出 light_count_keil_Y.Out1 等与实际硬件端口连接
    ...
}
```

```
int_T main();                          //删除不必要的输入参数
int_T main()                           //删除不必要的输入参数
{   light_count_keil_initialize();
    TMOD = 0x01;                       // 定时器 0 工作于16 位计时状态
    TH0 = (65535 - 50000)/256;         // 定时 0.05s 的高位初值
    TL0 = (65535 - 50000)%256;         // 定时 0.05s 的低位初值
    IE = 0x82;                         // 允许定时器 0 中断
    TR0 = 1;                           // 启动定时器
    light_count_keil_U.In1 = 0;        // 输入信号初始为 0
    while(1)
    {   if(num==1)                     // 满足定时标志时
        {   num = 0;                   // 定时标志清零
            light_count_keil_U.In1 = ~light_count_keil_U.In1;
        }                              // 翻转输入信号,实现 Stateflow 脉冲信号
        rt_OneStep();
    }
    //删除以下代码
    //printf("Warning: The simulation will run forever. "
    //"Generated ERT main won't simulate model step behavior. "
    //"To change this behavior select the 'MAT-file
    //logging' option.\n");
    //fflush((NULL));
...
}
```

代码修改完成后单击工具栏按钮，编译工程，如图 5.2.90 所示，窗口下部的信息显示已成功生成 HEX 文件。



图 5.2.90　编译生成.hex 文件

5. 虚拟硬件测试

根据本章第 5.1 节的介绍，建立 proteus 流水灯模型，并加载先前生成的 HEX 文件。单击“仿真”按钮，模型即以 0.1 s 为间隔，逐一亮灯，实现了 Simulink 模型所设计的功能，如图 5.2.91 所示。

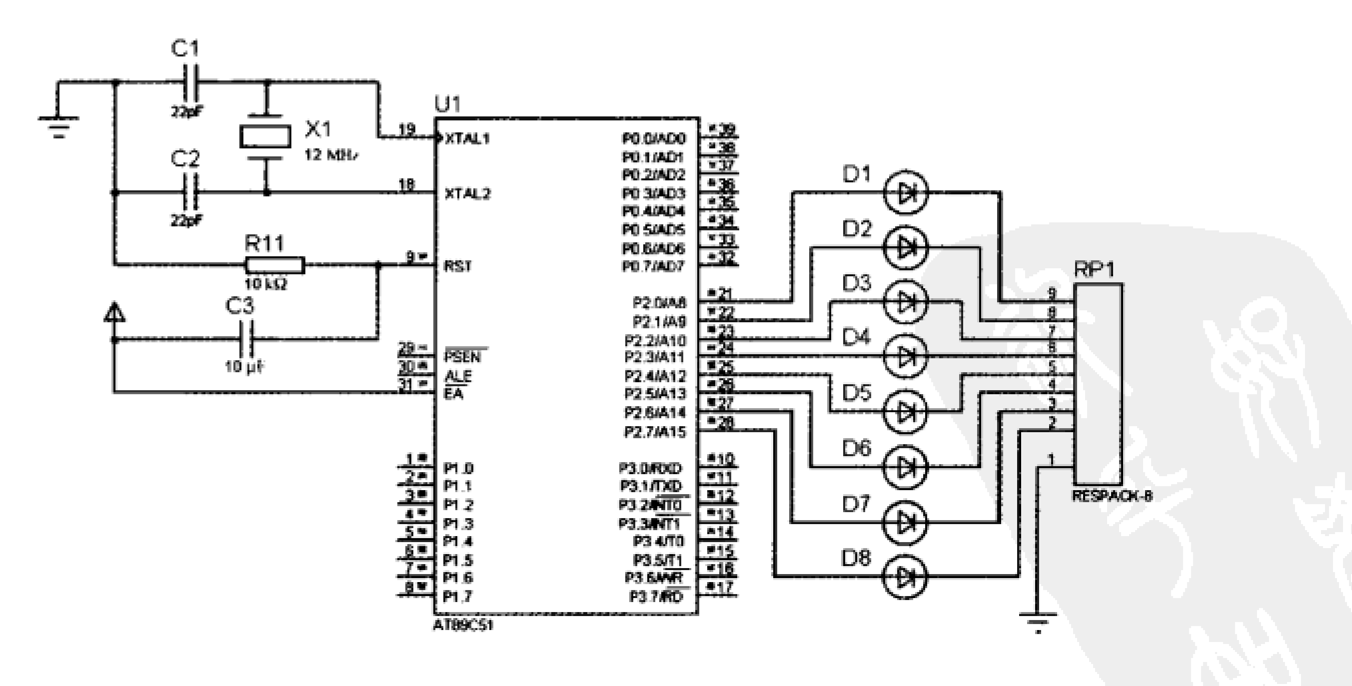
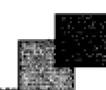


图 5.2.91　虚拟硬件测试结果



5.3 TASKING 嵌入式开发环境(EDE)

TASKING 是一家专业开发嵌入式系统软件工具的公司，1974 年创建于荷兰，2001 年并入 Altium，其产品仍称作 TASKING。TASKING 的交叉式编译工具、调试器、集成开发环境(IDE)、实时操作系统(RTOS)，广泛支持各种嵌入式领域的 DSP、8－16－32 位处理器与微控制器。读者在本节末尾将看到，对于同样一段 C 代码，使用 TASKING 编译器可以得到比 Keil 更高的编译效率。

5.3.1 预备知识

1. 软件下载与安装

MathworksR2010b 支持的 TASKING 版本如表 5.3.1 所列。

表 5.3.1　MathworksR2010b 支持的 Tasking 版本

处理器	编译器版本
Infineon® TriCore®	TASKING VX-toolset for TriCore v2.5 r2
Infineon® C166®	TASKING Tools for C166/ST10 v8.7 r1
Renesas M16C	TASKING Tools for M16C v3.1 r1 patch 2
ARM®	TASKING C Compiler for ARM v2.0 r2
Freescale DSP563xx	TASKING Tools for DSP563xx v3.5 r3 patch 2
8051	TASKING Tools for 8051 v7.2 r1

用户若没有对应版本的软件，可以径自前往：

http://www.tasking.com/products/trials-and-demos.shtml，下载试用版软件。例如单击链接 8051 Trial Version，下载 8051 编译器，如图 5.3.1 所示。

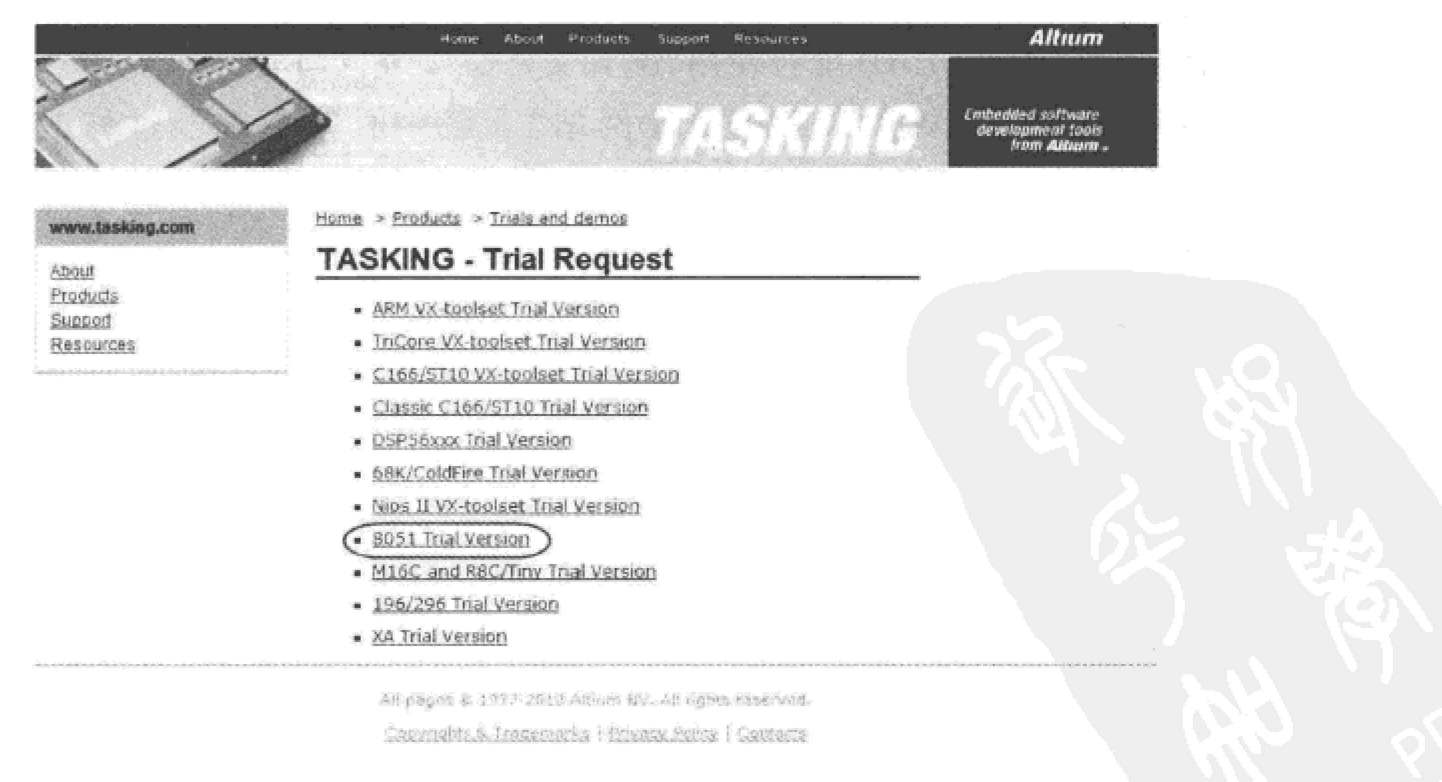


图 5.3.1　试用版软件下载页面

在后续打开的注册页面里输入必要信息，按下右下方的 Continue，即可下载（图 5.3.2）。

Home About Products Support Resources Altium

TASKING Embedded software development tools from Altium

www.tasking.com
About
Products
Support
Resources

Home
TASKING - 8051 toolset Trial Version Request

Contact Details

First Name | Last Name
Email
Department | Job title
Phone | Fax
Company
Address
City | State/Province
Zip
Country/Region | Please Select

I would like to:
Download the 8051 toolset Trial Version files

Continue

图 5.3.2　注册页面

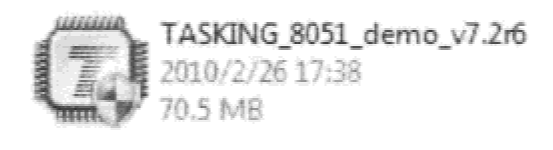


图 5.3.3　Tasking 8051 编译器 v7.2r6

下载得到 8051 编译器如图 5.3.3 所示，不过该试用版软件仅作为学习之用，如果用于实际项目开发，为避免不必要的限制，请读者使用完全版的软件。

按提示安装完成，即可作为常规的编译环境使用。不过一般来说，目前可供下载的试用版本与表 5.3.1 所列的不相符，由于 MATLAB 的配置文件仅针对默认版本，无兼容性可言，为了当前的 TASKING 软件能与 MATLAB 配合使用，配置文件必须进行修改。

请用户登录北京航空航天大学出版社网站“下载中心”，下载经过修改的 EDE 与 pjt 文件。

2. TASKING EDE 的使用

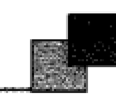
(1) 打开 TASKING EDE [8051]，选择菜单项 File→Change Directory，设置当前工作目录，例如…\test，如图 5.3.4、图 5.3.5 所示。

(2) 选择菜单项 File→New Project Space…，创建新的工程空间，在 Filename 栏中输入工程空间名（如 Hello World），如图 5.3.6 所示。

创建后的工程空间里，尚不包含任何工程，如图 5.3.7 所示。

(3) 在工程属性对话框中，单击按钮，为该工程空间新增一个工程，名为 Hello World，如图 5.3.8 所示。

创建的新工程里无任何代码文件，如图 5.3.9 所示。



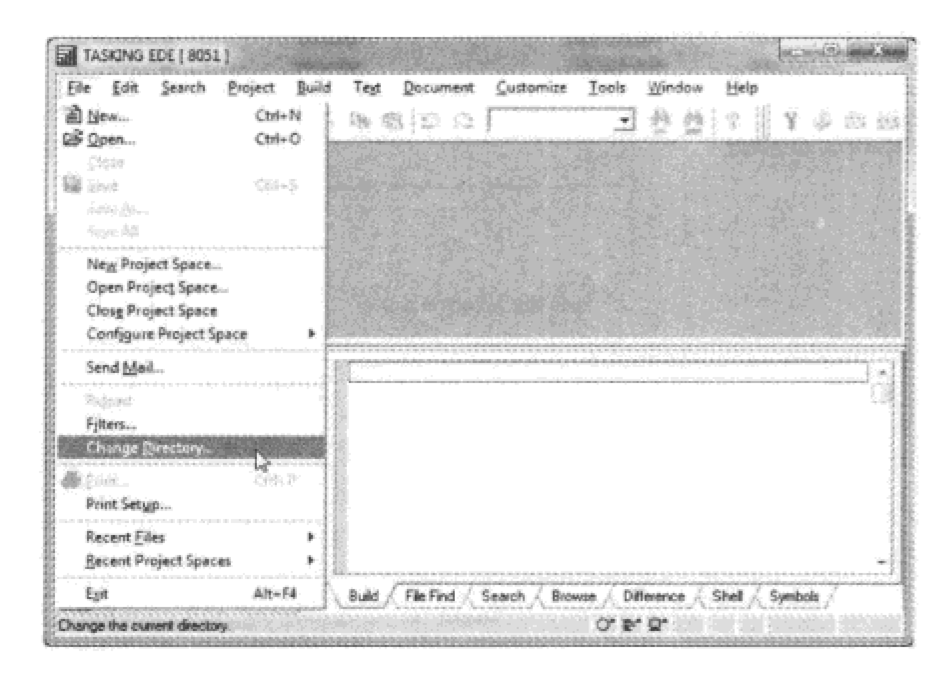


图 5.3.4 设置工作目录菜单

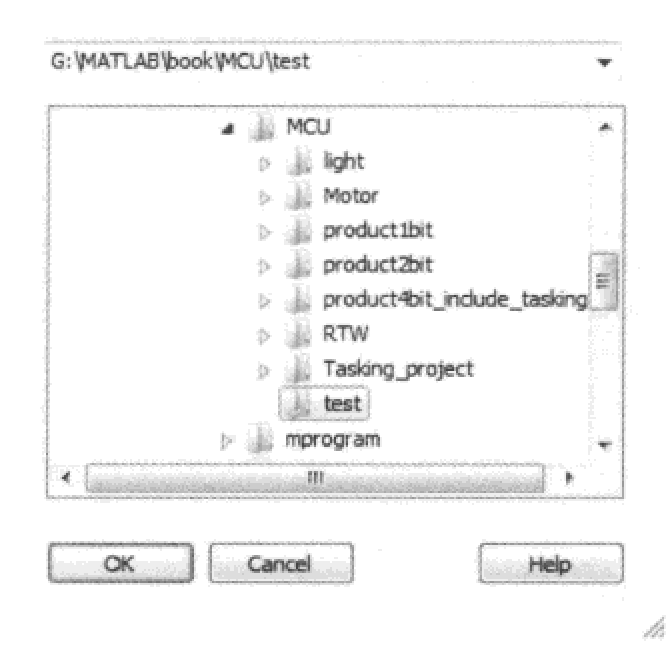


图 5.3.5 设置工作目录

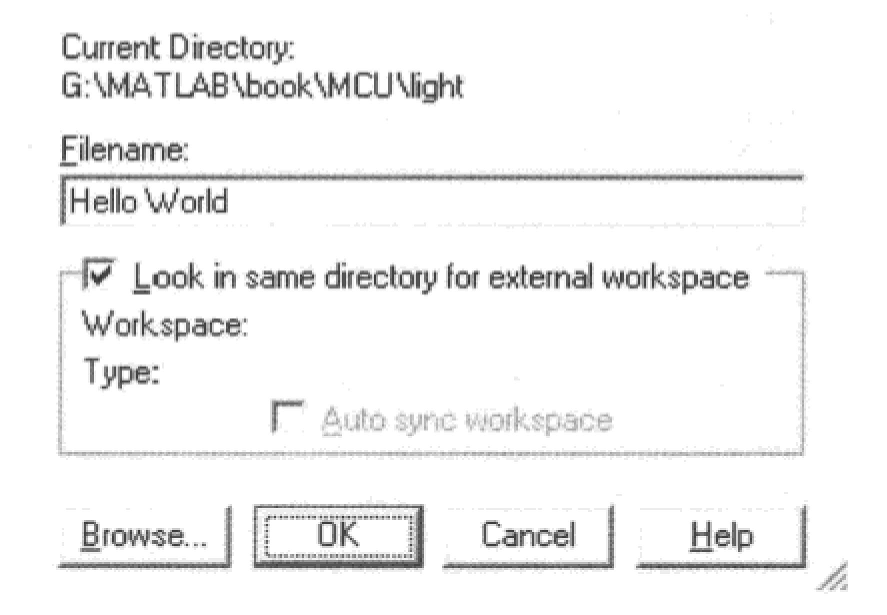


图 5.3.6 创建新的工程空间

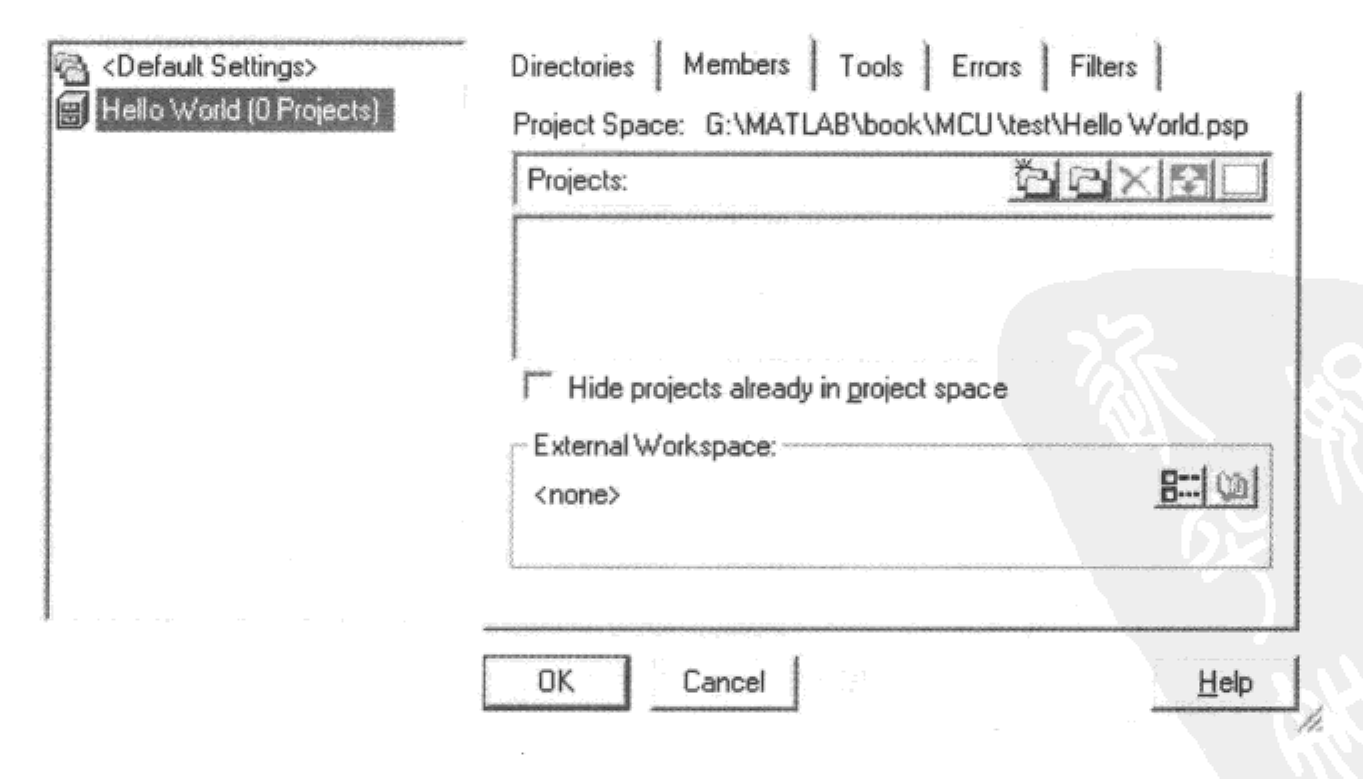


图 5.3.7 工程空间属性

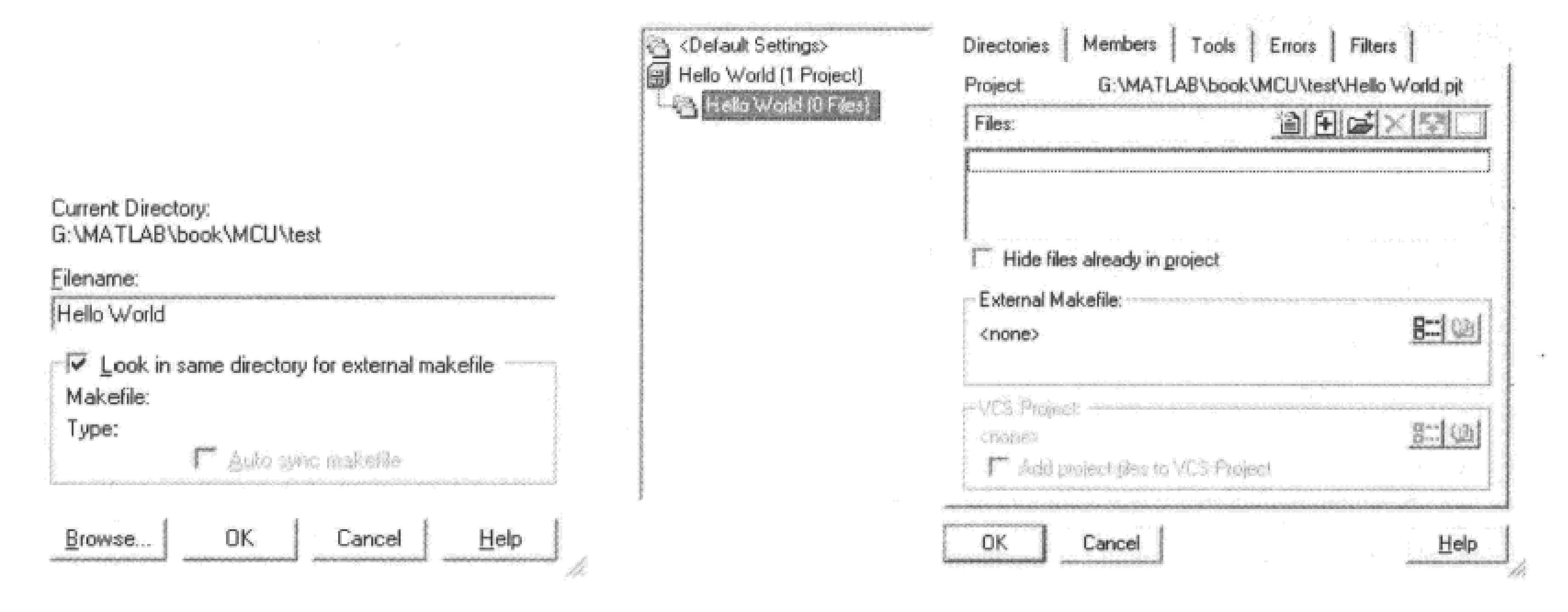


图 5.3.8　添加新的工程　　　　图 5.3.9　工程空间属性

(4) 单击图标，为当前工程新增一个源代码文件，例如 hello. c，如图 5.3.10 所示。

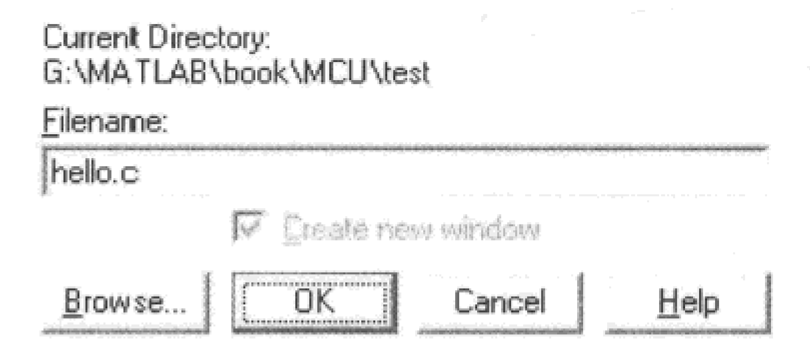


图 5.3.10　添加源代码文件

这时工程属性对话框的文件列表即显示该文件的路径，如图 5.3.11 所示。

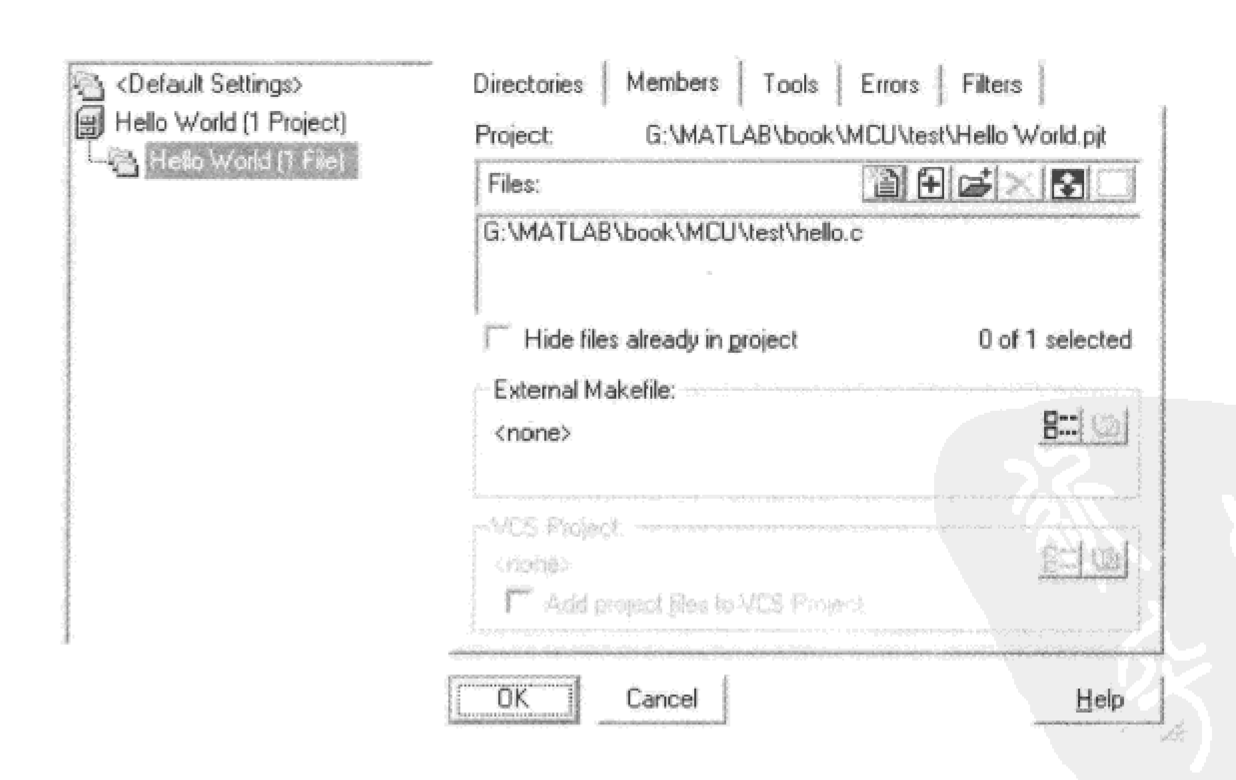
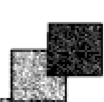


图 5.3.11　工程空间属性

在随后打开的编辑窗口中添加以下代码，如图 5.3.12 所示。



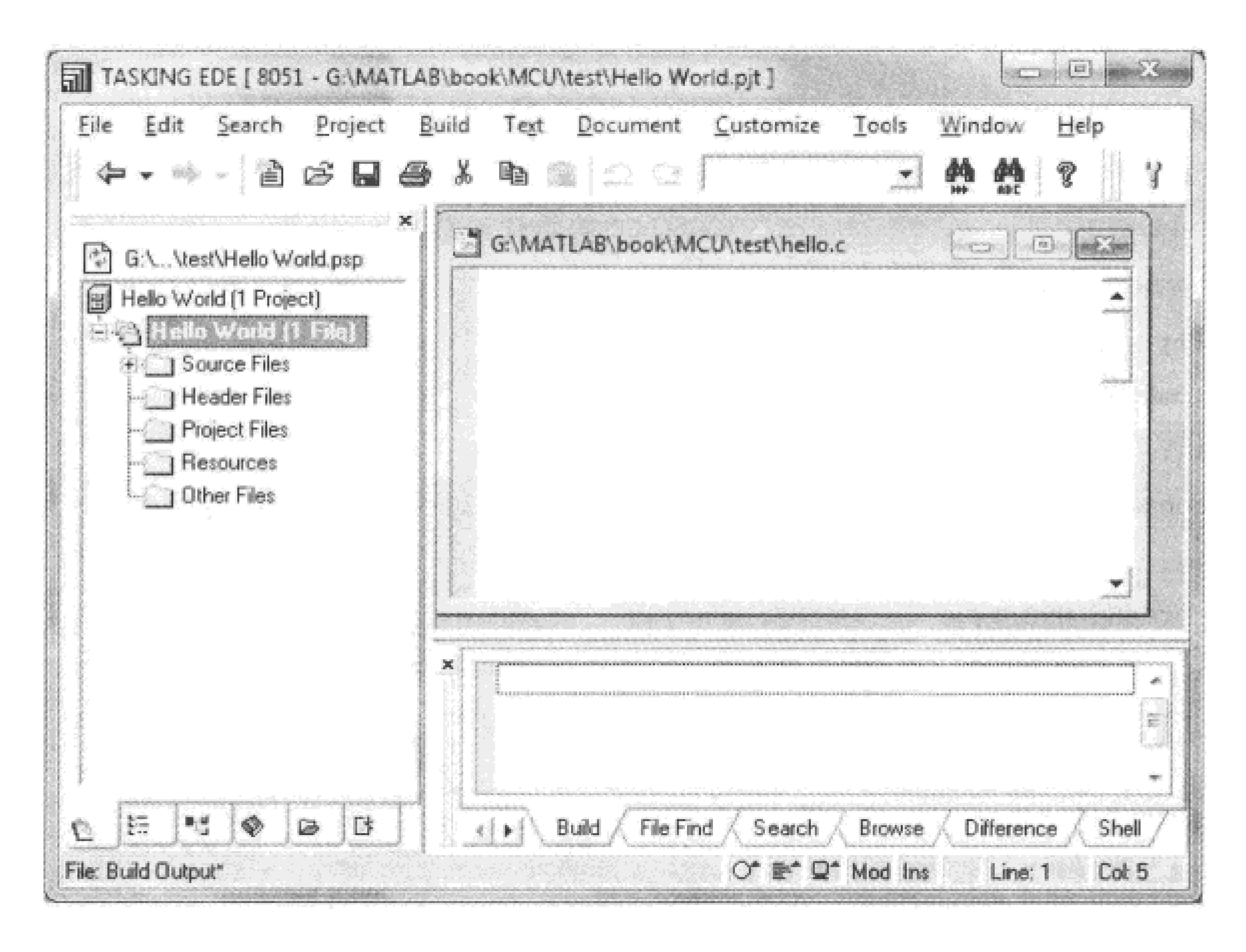
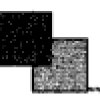


图 5.3.12　编辑源代码

```
#define uchar unsigned char
#define uint unsigned int
uchar t1[] = "Hello World! ";          //定义 LCD 显示的字符
void delay(uint x)                      //延时函数
{  uchar y;
   while(x--)
   {  for(y = 0;y < 100;y + + );
   }
}
void write_command(uchar cmd)           //LCD 命令写入函数
{  P2_0  =  0;
   P2_1  =  0;
   P2_2  =  0;
   P0  =  cmd;
   P2_2  =  1;
   delay(1);
   P2_2  =  0;
}
void write_data(uchar dat)              //数据写入函数
{  P2_0  =  1;
   P2_1  =  0;
   P2_2  =  0;
   P0  =  dat;
   P2_2  =  1;
```




```
    delay(1);
    P2_2 = 0;
}
void initialize()                          //LCD 初始化函数
{   write_command(0x38);
    delay(1);
    write_command(0x01);
    delay(1);
    write_command(0x06);
    delay(1);
    write_command(0x0f);
    delay(1);
}
void main()
{   uint i;
    initialize();
    write_command(0x81);                   //显示"Hello World! "
    for(i=0;i<12;i++)
    {   write_data(t1[i]);
        delay(200);
    }
    while(1);
}
```

(5) 单击 EDE 主窗口的按钮 ，打开工程选项窗口，在 Processor→Processor Selection 面板，在下拉菜单 8051 CPU Name 中选择 AT89C51 选项，如图 5.3.13 所示。

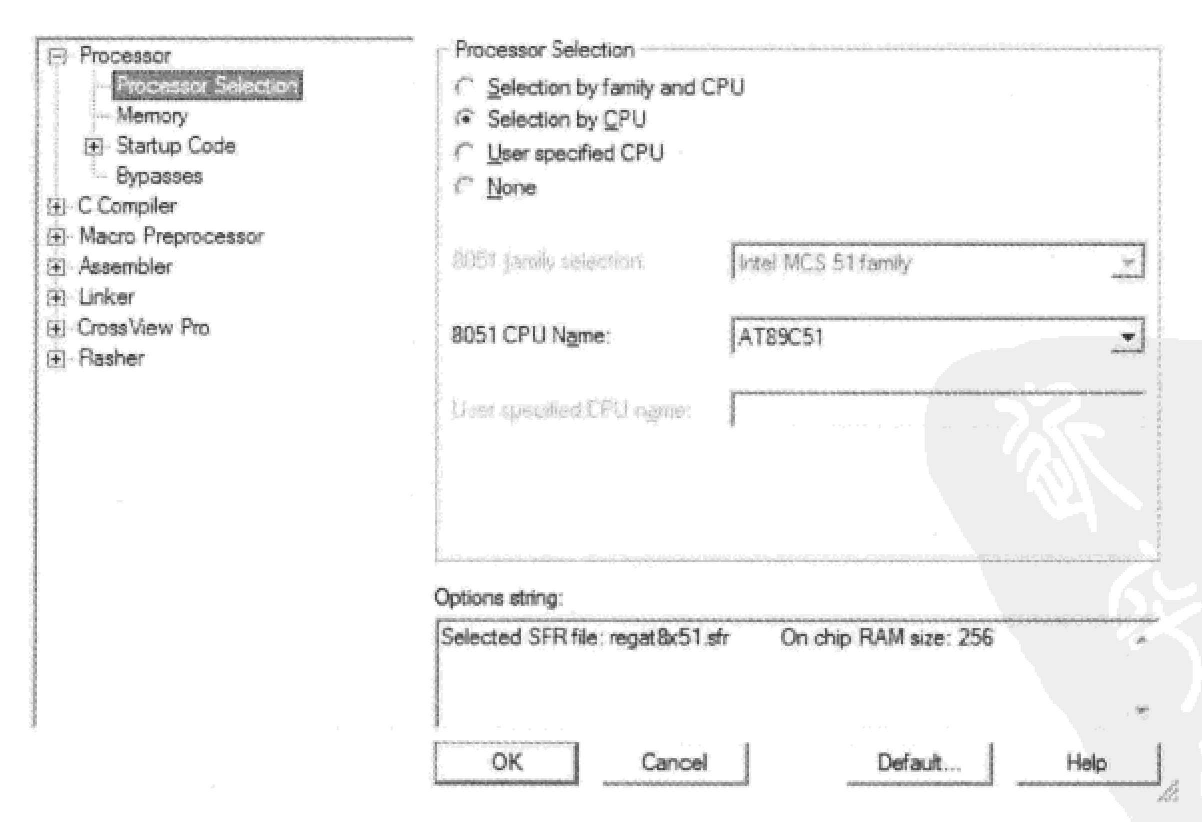
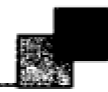


图 5.3.13　选择处理器



在 Linker→Output Format 面板，勾选 Intel Hex records for EPROM programmers(.hex)复选框，如图 5.3.14 所示。

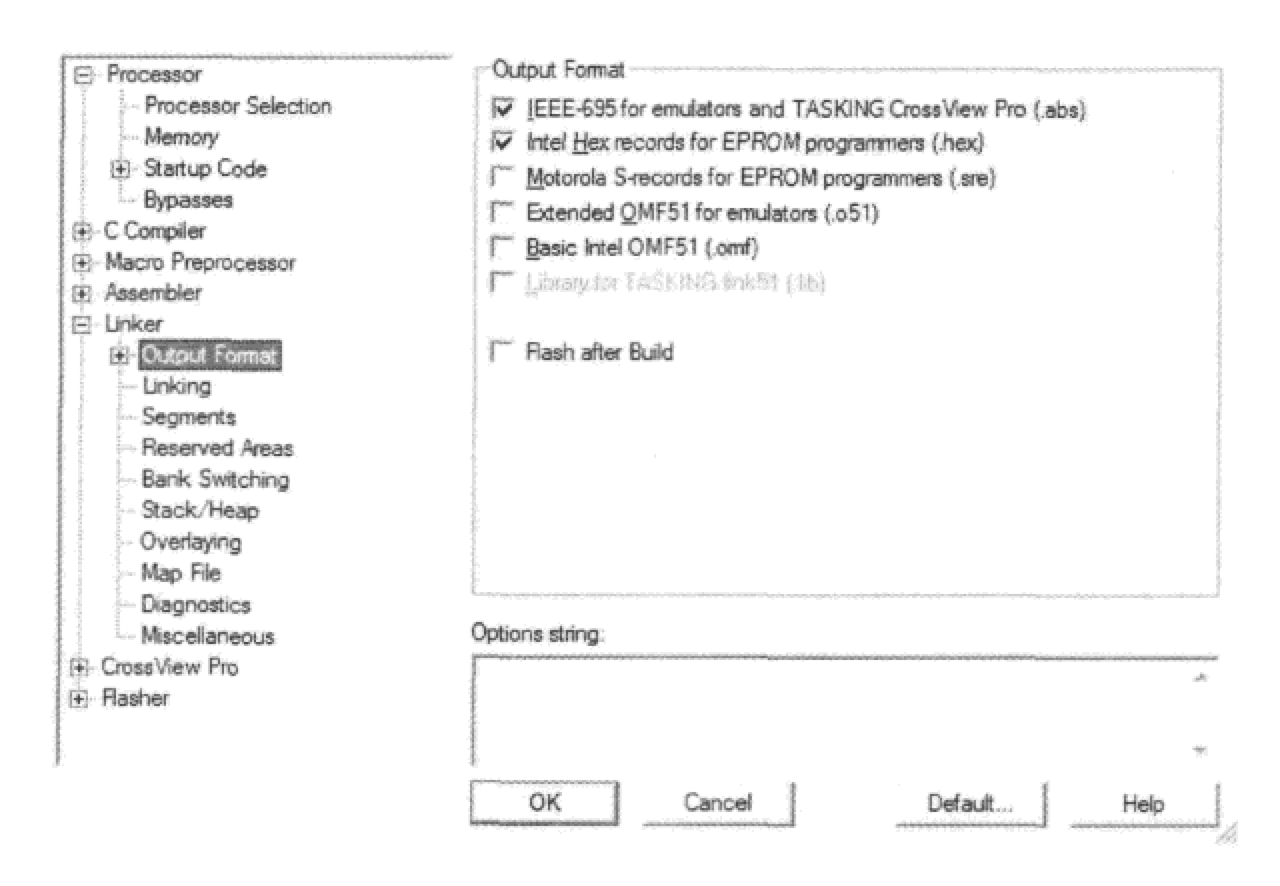


图 5.3.14　选择输出文件类型

(6) 单击工具栏按钮编译工程，窗口下部的信息显示已成功生成 HEX 文件，如图 5.3.15所示。

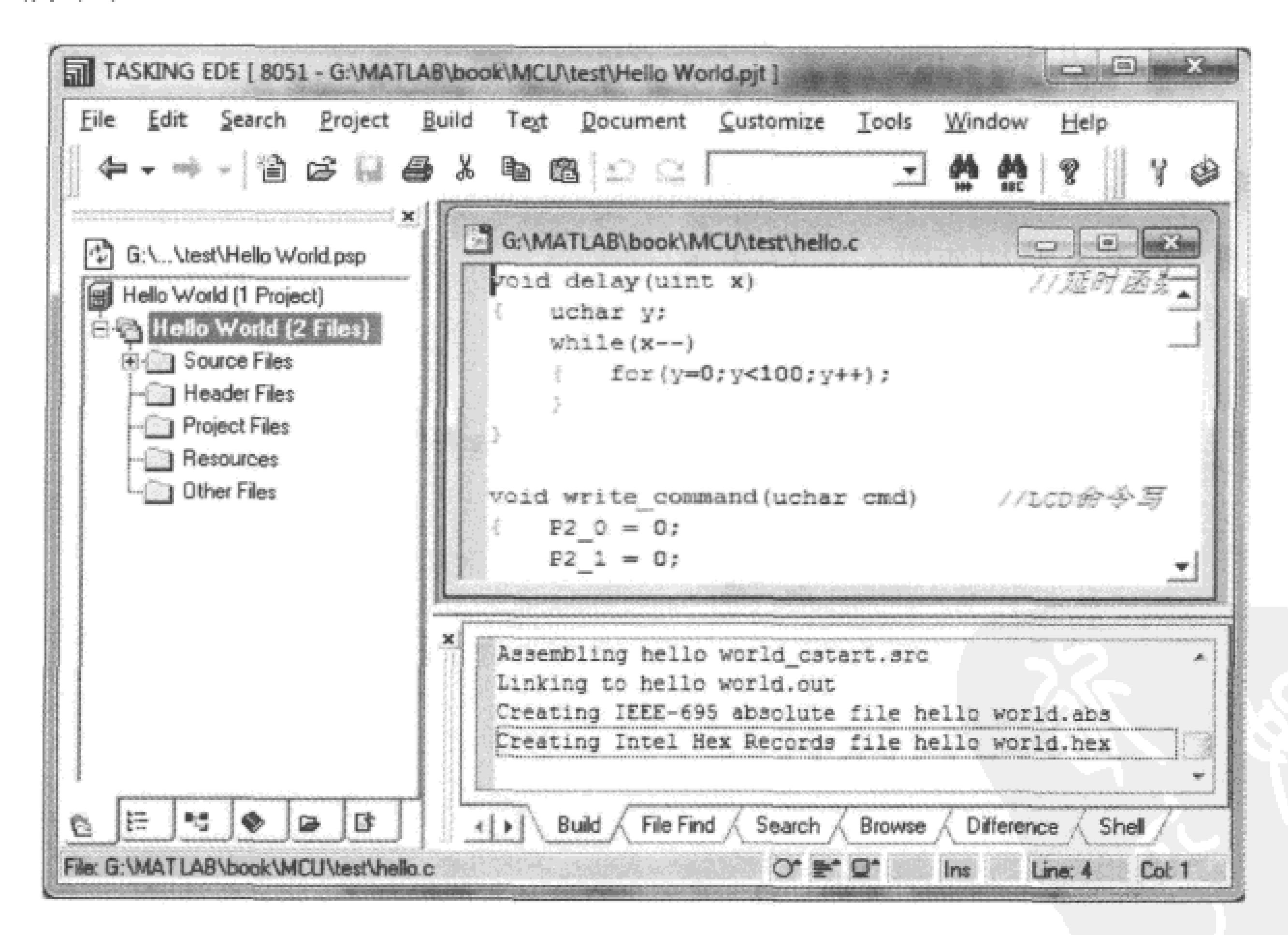


图 5.3.15　编译生成 HEX 文件

(7) 根据第 5.1 节的介绍，建立 proteus 模型，并加载先前生成的 HEX 文件，单击“仿真”按钮，LCD 屏幕显示 Hello World，实现了 C 代码的功能，如图 5.3.16 所示。

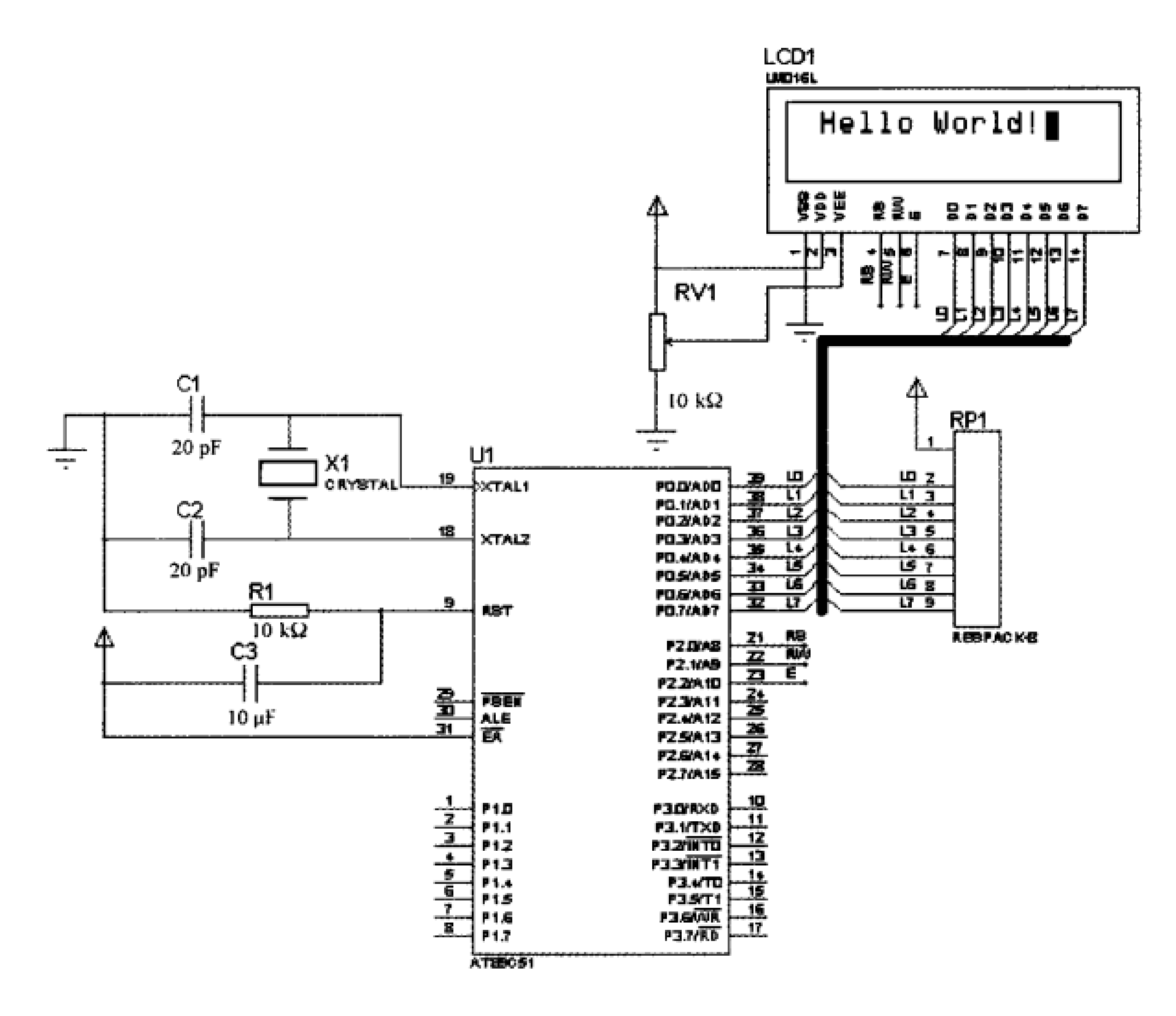


图 5.3.16 虚拟硬件测试结果

5.3.2 直流电动机控制

1. 直流电动机原理

直流电动机的基本结构如图 5.3.17 表示。

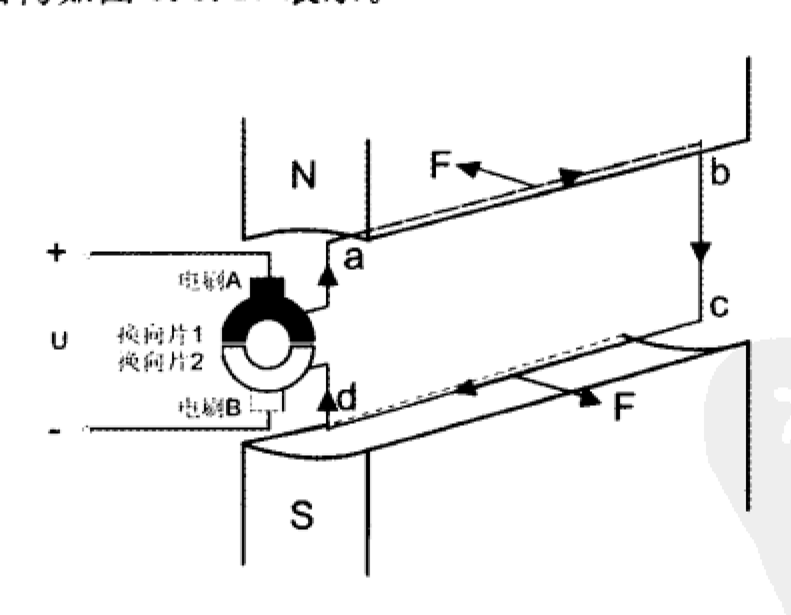
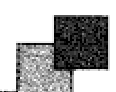


图 5.3.17 直流电动机基本结构

AB 两电刷间加上正下负的直流电压时，在图 5.3.17 所示的瞬间，电流的方向为：电刷 A→换向片 1→a→b→c→d→换向片 2→电刷 B，根据左手定则，导体 ab 受到向左的电磁力，导体 cd 受到向右的电磁力，电磁力对转轴形成逆时针方向的转矩，于是转子向逆时针方向旋转。



旋转 180°后(图 5.3.18),换向片 2 与电刷 B 接触,换向片 1 与电刷 A 接触,电流的方向为:电刷 A→换向片 2→d→c→b→a→换向片 1→电刷 B,根据左手定则,导体 cd 受到向左的电磁力,导体 ab 受到向右的电磁力,电磁力对转轴依旧形成逆时针方向的转矩,于是转子持续向逆时针方向旋转。

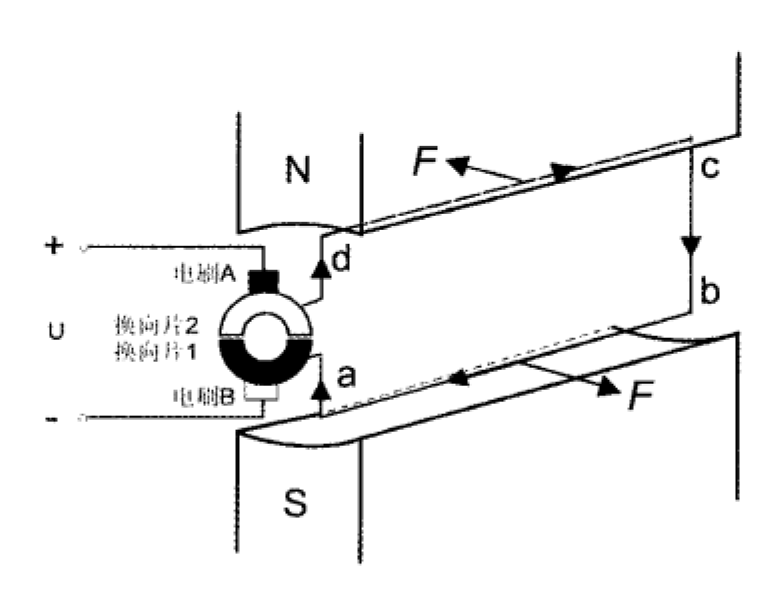


图 5.3.18　直流电动机基本结构

加载在电刷之间的电压方向,决定了电动机的旋转方向,为了能方便地利用单片机控制电动机旋转,可以在两者之间加入一块驱动芯片 L298。L298 是一款高电压、大电流双全桥式驱动芯片,最大工作电压 46V,总直流电输出 4A,内部原理如图 5.3.19 所示。它可接收标准的 TTL 逻辑信号,驱动感性负载,例如伺服电动机、继电器、螺线管、直流电动机以及步进电动机,芯片提供了两个使能端 EnableA 与 EnableB,可分别独立控制 Output1、2 及 Output3、4 所连接的设备。

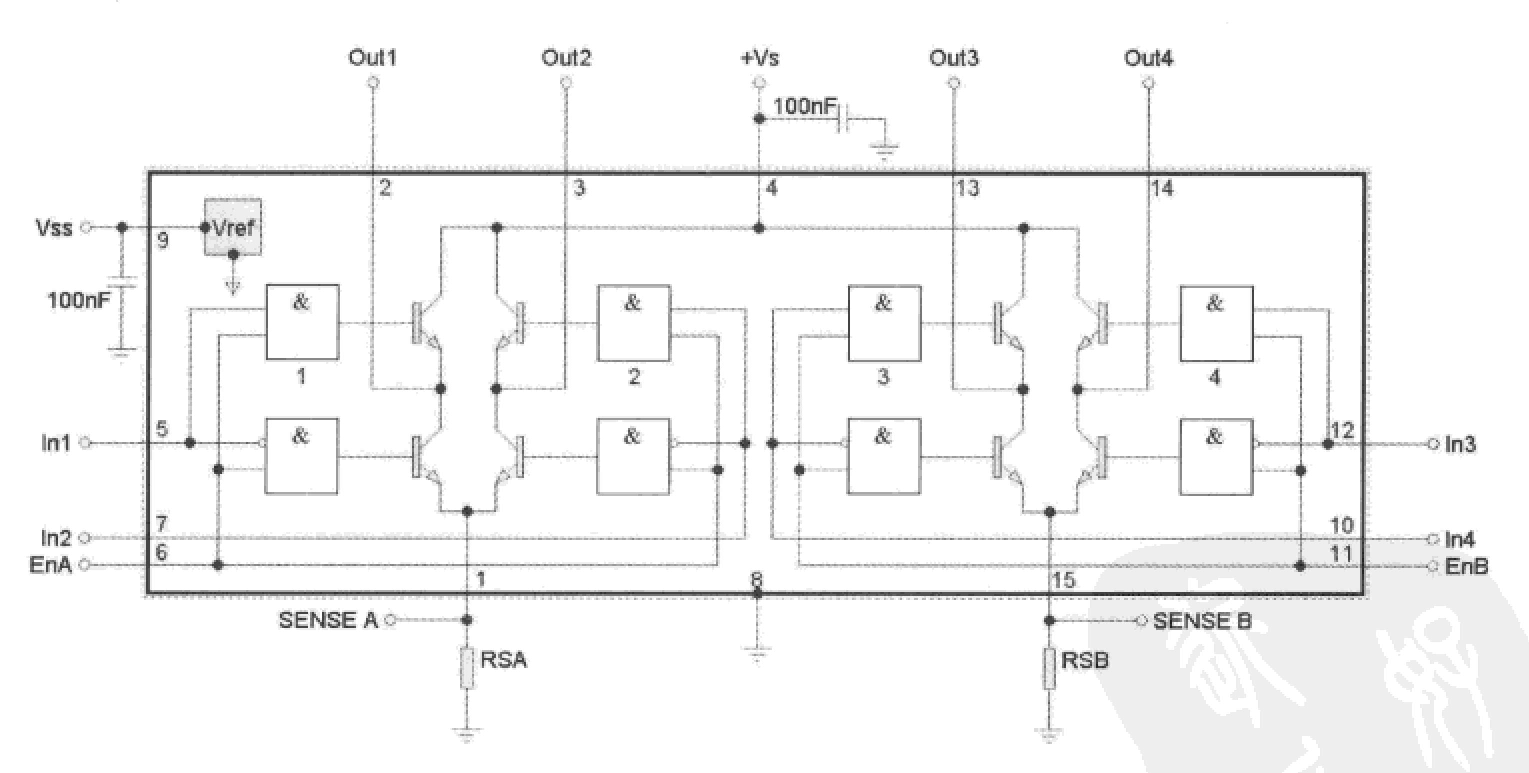


图 5.3.19　L298 内部原理

以原理图左侧的全桥为例,若直流电动机的两个电刷分别接在 Out1 与 Out2 引脚,当 Input1、Input2、EnableA 的输入为表 5.3.2 左列时,电动机即按右列的情况开始旋转或停止(电路的实际连接,请参考下文虚拟硬件测试)。

表 5.3.2 L298 控制逻辑

输 入		功 能
EnableA＝1	Input1＝1，Input2＝0	正转
	Input1＝0，Input2＝1	反转
	Input1＝Input2	快速停转
EnableA＝0	Input1＝x，Input2＝x	自由停转

2. 直流电动机控制状态图

由于本书使用的 Tasking EDE 是试用版，有代码量限制，因此根据上述电动机控制原理以及第 3 章的介绍，建立简易的直流电动机控制 Stateflow 状态图，如图 5.3.20 所示，实现电动机的正转、反转与快速停转的控制逻辑（其中 EnableA 在下文的代码修改时，永久置为高电平）。

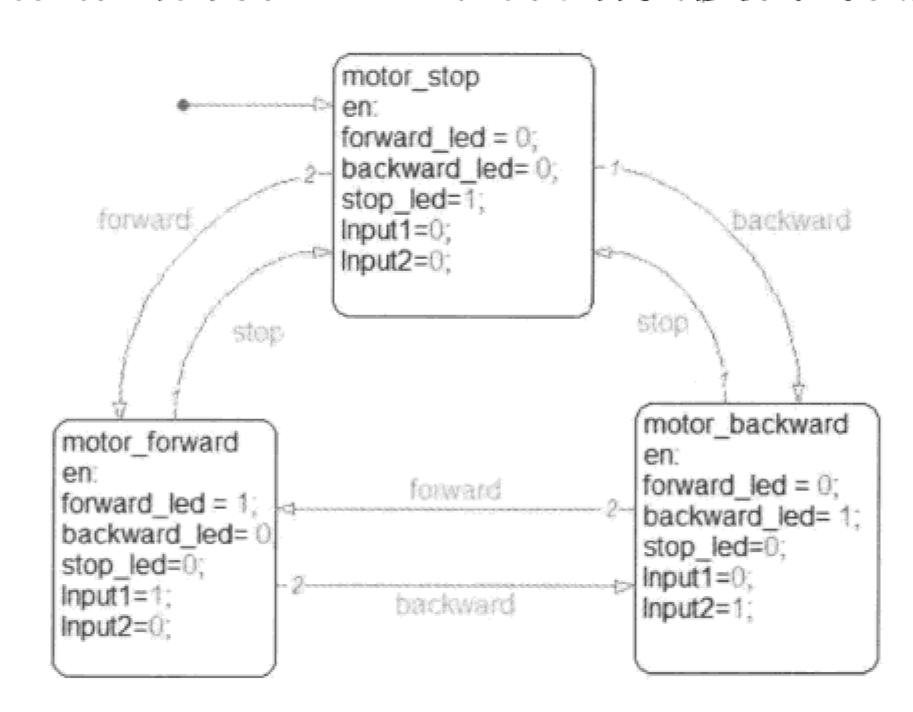


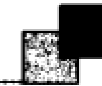
图 5.3.20 直流电动机控制状态图

其中数据 forward_led、backward_led、stop_led 作为电动机正转、反转、停转的指示灯信号，数据 Input1、Input2 作为电动机驱动芯片 L298 的正反转输入信号。

事件 forward、backward、stop 作为电动机运转的控制信号（下降沿触发或根据实际需要），事件 tic 作为激活 stateflow 状态图的信号。数据及事件列表如图 5.3.21 所示。

Name	Scope	Port	Resolve Signal	DataType	Trigger	Compi
forward	Input	1			Falling	
backward	Input	2			Falling	
stop	Input	3			Falling	
tic	Input	4			Either	
forward_led	Output	1		double		
backward_led	Output	2		double		
stop_led	Output	3		double		
Input1	Output	4		double		
Input2	Output	5		double		

图 5.3.21 状态图数据及事件列表



3. 功能验证模型

完成 Stateflow 状态图之后，在 Simulink 模块库中找到图 5.3.22～图 5.3.24 所示的模块。

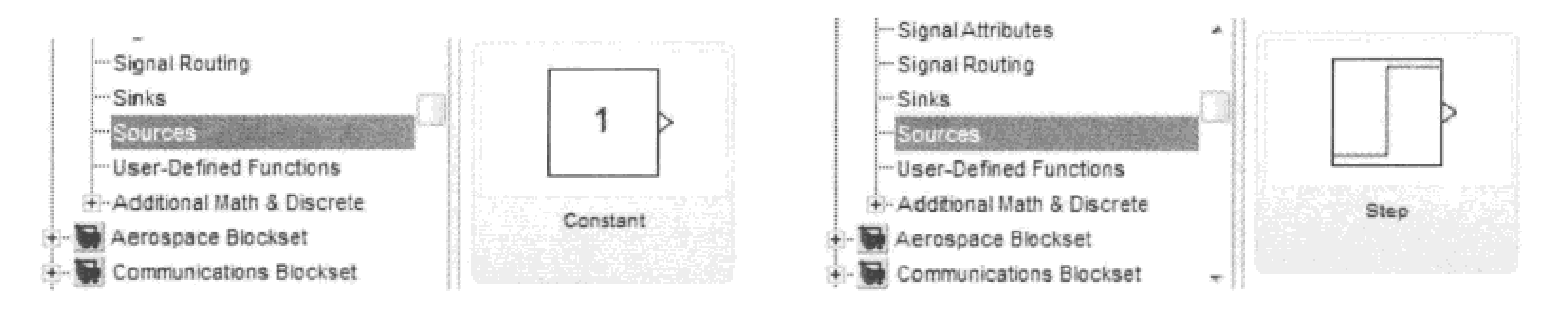


图 5.3.22　Constant 模块　　　　图 5.3.23　Step 模块

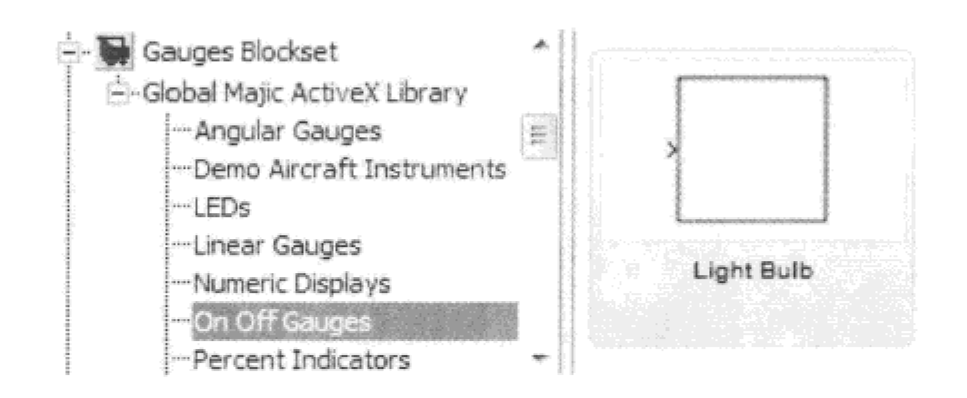


图 5.3.24　Light Bulb 模块

根据第 3 章的介绍建立 GUI 界面，并连接各模块，如图 5.3.25 所示。

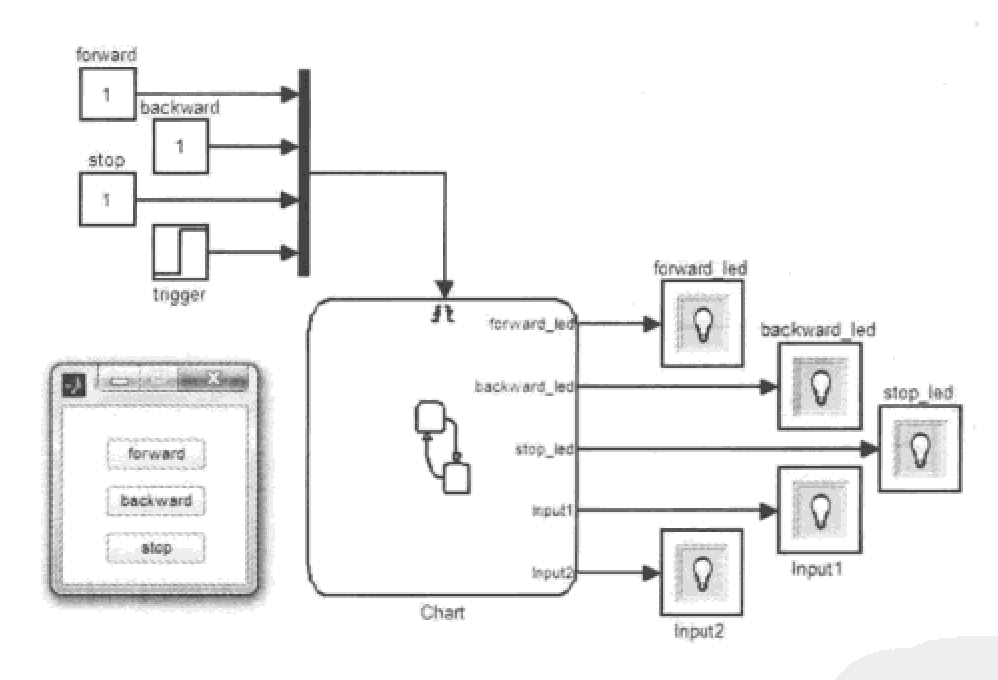


图 5.3.25　功能验证模型

选择模型主窗口的菜单项 Simulation→Configuration Parameters…，打开模型参数对话框，在 Solver options 选项区域，设置求解器为定步长离散求解器，步长为 0.01，如图 5.3.26 所示。

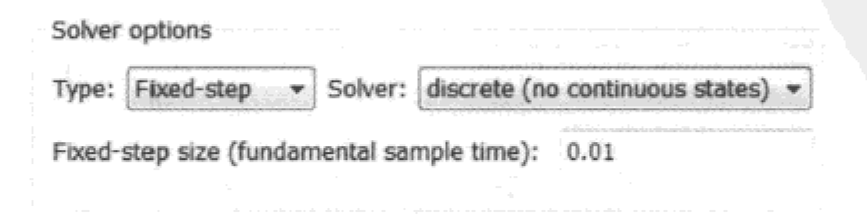


图 5.3.26　求解器设置

双击阶跃信号模块，设置阶跃时间为采样时间的整数倍，例如 0.2，如图 5.3.27 所示。

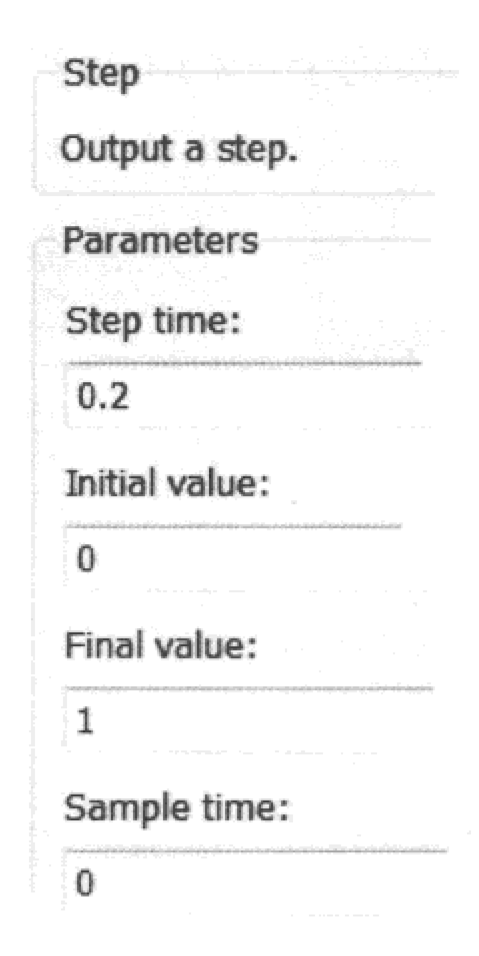


图 5.3.27　设置 Step 模块的阶跃时间

单击模型仿真按钮，再分别单击 GUI 界面的正转、反转、停转按钮，LED 灯即依次亮起，实现了 Stateflow 所描述的功能，如图 5.3.28～图 5.3.30 所示。

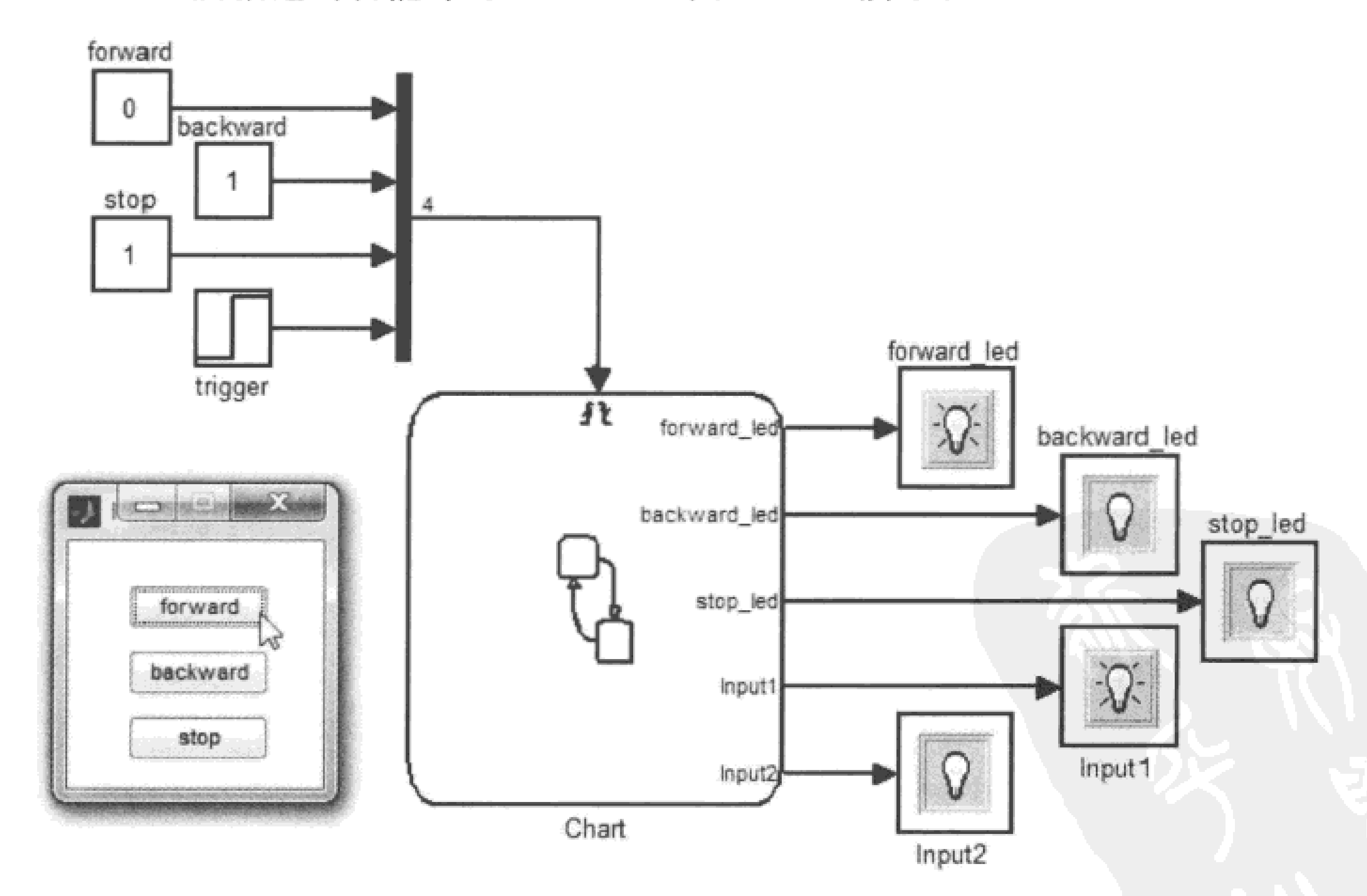
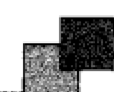


图 5.3.28　仿真结果：正转



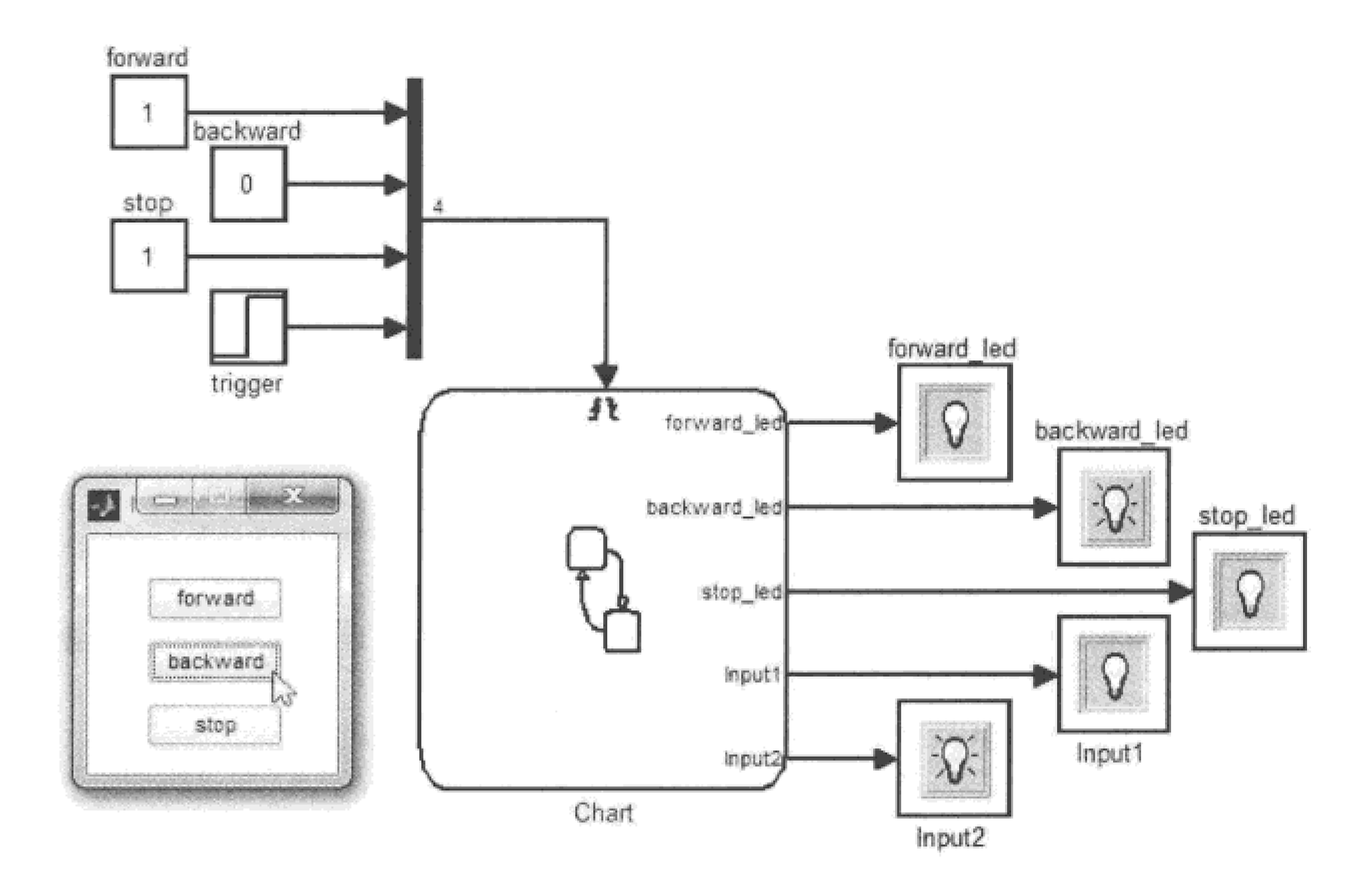


图 5.3.29　仿真结果:反转

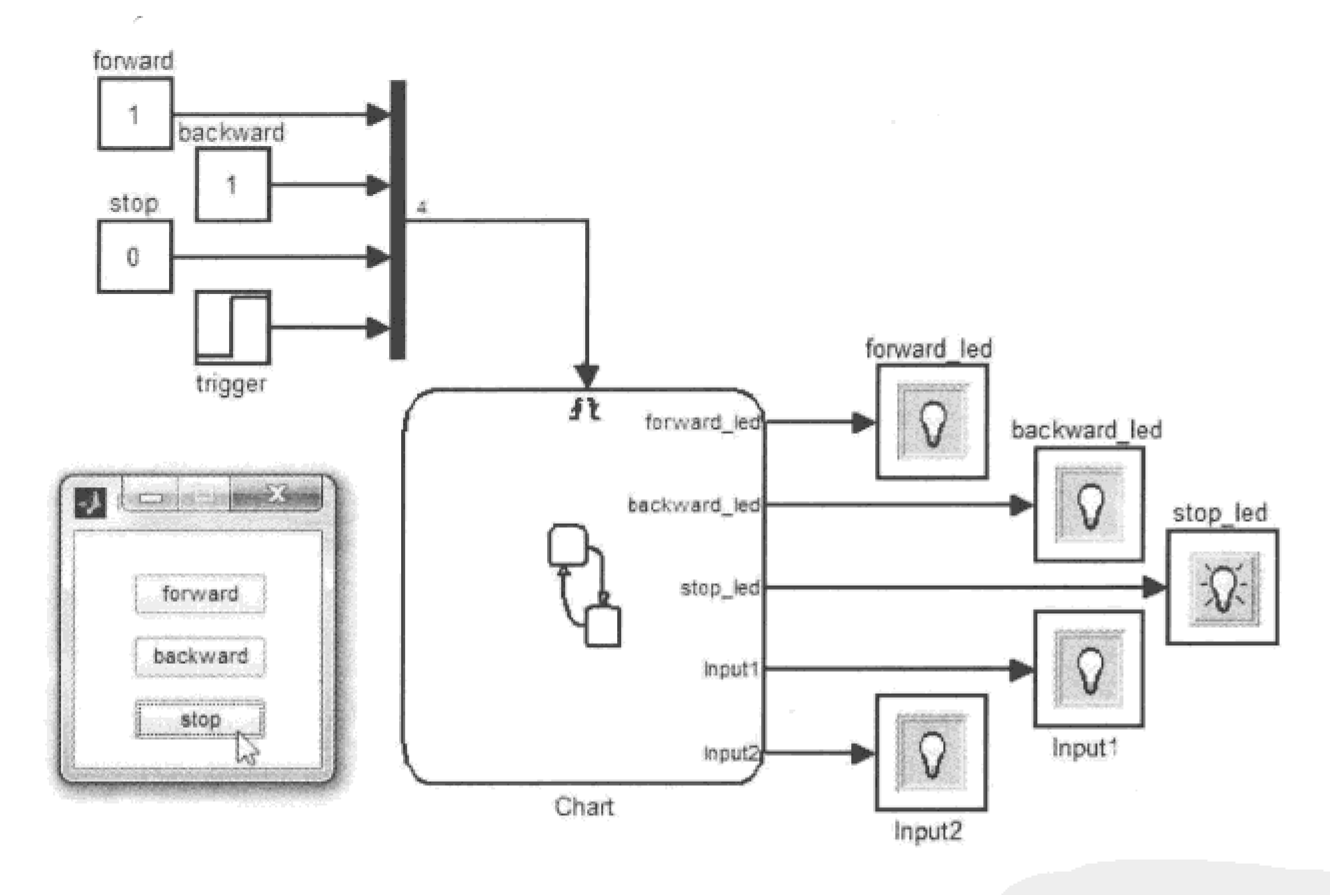


图 5.3.30　仿真结果:停转

4. 软件在环测试

(1) 数据类型转换。在模块库 Simulink→Ports & Subsystems 中找到输入模块与输出模块,替换图 5.3.25 中的常数、阶跃与 LED 模块,并将模型另存。

单击模型窗口的按钮,打开模型浏览器,将电动机控制状态图里各变量的数据类型设置为 uint8,如图 5.3.31 所示。

功能验证模型中的 In 模块的数据类型同样设置为 uint8,Out 模块的数据类型可设为自动继承,也可强制设置为 uint8,如图 5.3.32 所示。

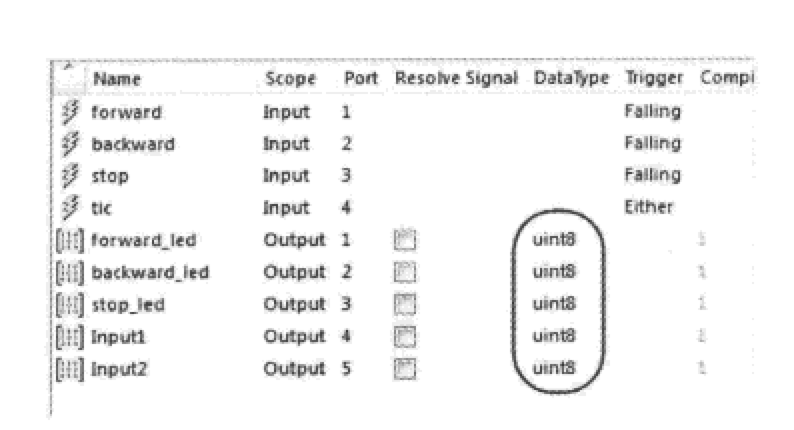


图 5.3.31　设置模型状态图中的数据类型

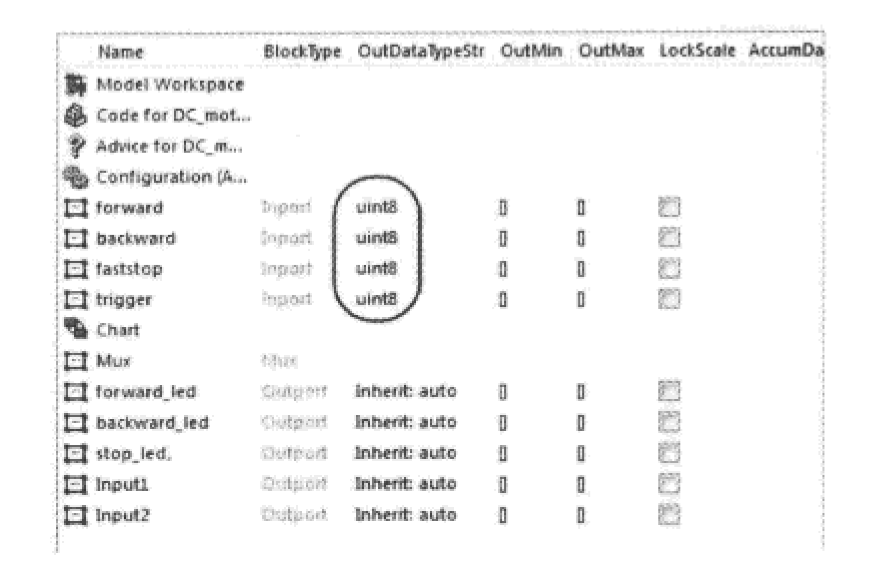


图 5.3.32　设置模型端口的数据类型

修改后的代码生成模型如图 5.3.33 所示。

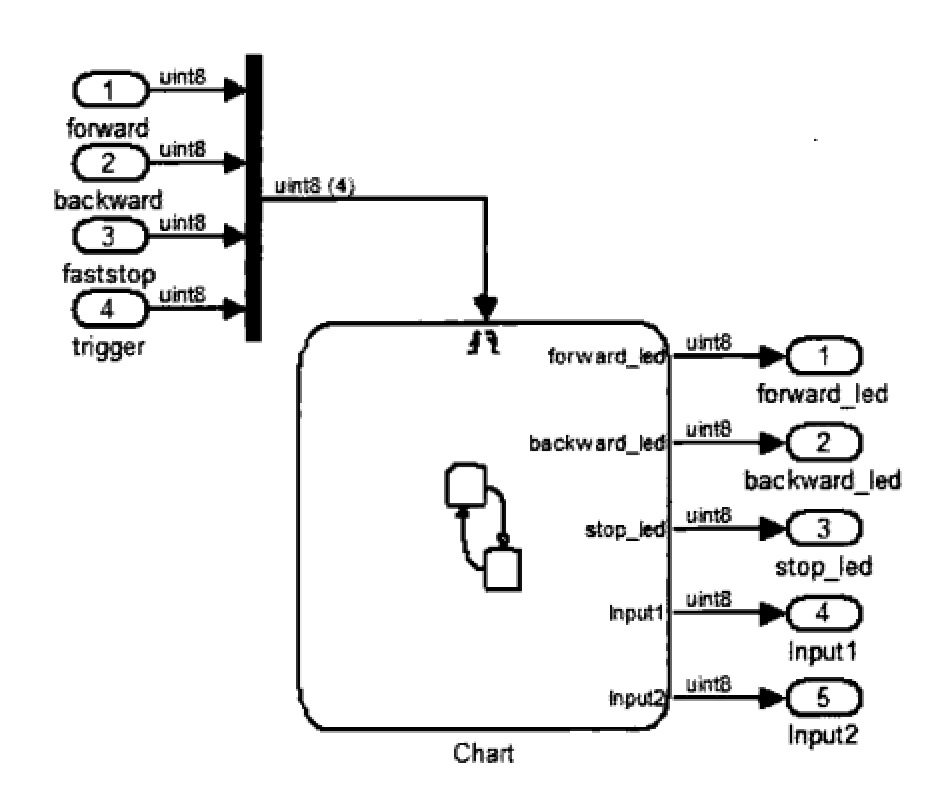


图 5.3.33　代码生成模型

(2) 模型参数设置。打开模型参数对话框，在 Real-Time Workshop 面板设置 TLC 文件为 ert.tlc，如图 5.3.34 所示。

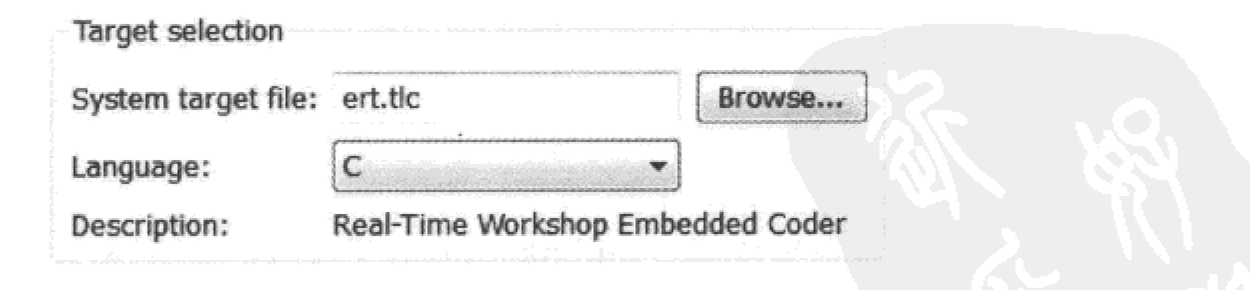
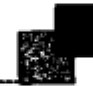


图 5.3.34　设置 TLC 文件

在 Real-Time Workshop→Interface 面板中，取消勾选不必要的复选框，如图 5.3.35 所示。

在 Real-Time Workshop→Report 面板中，勾选所有复选框，便于后期检查及跟踪，如图 5.3.36 所示。



Software environment

Target function library: C89/C90 (ANSI)

Utility function generation: Shared location

Support: floating-point numbers　non-finite numbers　complex numbers

absolute time　continuous time　non-inlined S-functions

variable-size signals

Multiword type definitions: System defined

图 5.3.35　Interface 面板

Create code generation report　Launch report automatically

Navigation

Code-to-model

Model-to-code　Configure...

Traceability Report Contents

Eliminated / virtual blocks

Traceable Simulink blocks

Traceable Stateflow objects

Traceable Embedded MATLAB functions

图 5.3.36　生成报告设置

(3) 生成 SIL 模块。在 Real-Time Workshop→SIL and PIL Verification 面板中的 Create block 下拉列表，选择 SIL 选项，如图 5.3.37 所示。

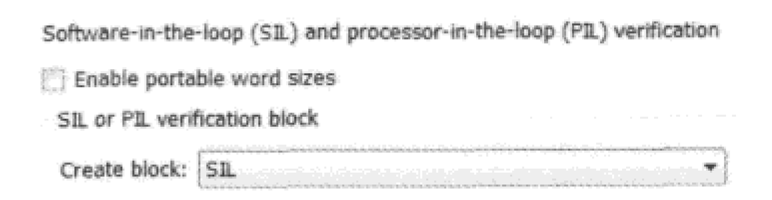


图 5.3.37　选择 SIL 选项

之后单击模型工具栏的按钮，得到代码生成报告与 SIL 模块，如图 5.3.38 和图 5.3.39 所示。

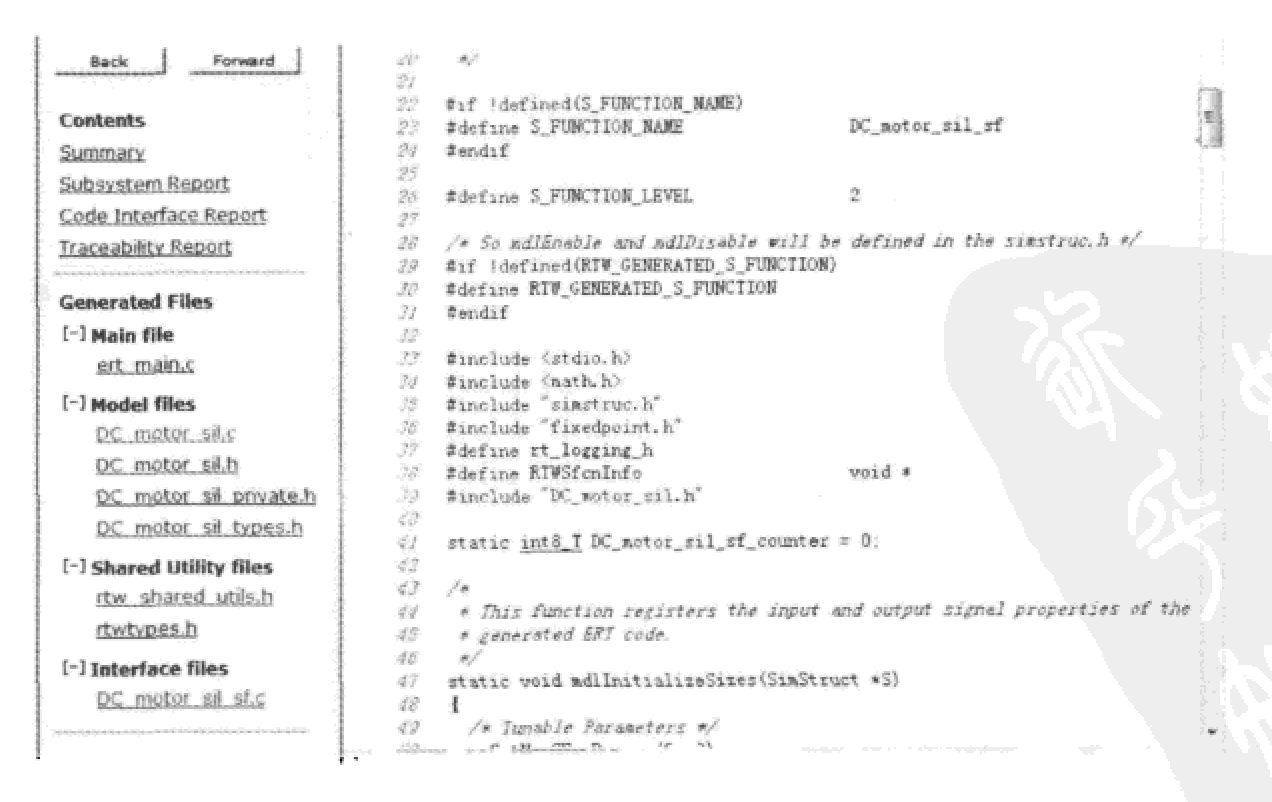


图 5.3.38　代码报告

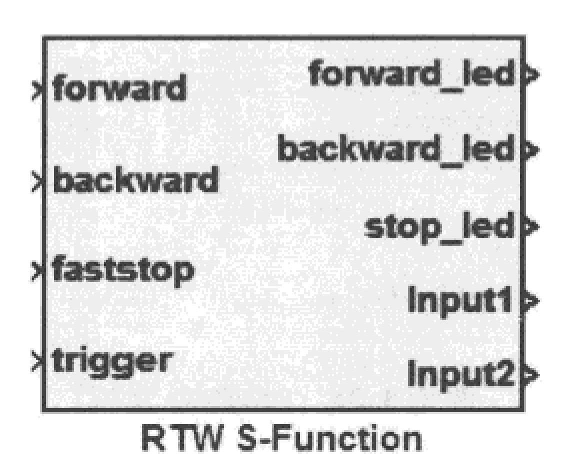


图 5.3.39 SIL 模块

以 SIL 模块替换图 5.3.25 的 Stateflow 模块，重建验证模型，并在各端口间加入必要的数据类型转换模块。

若用户仍计划使用 GUI 界面进行 SIL 测试，则必须确保各个输入模块（常数与阶跃模块）的采样时间为 0，如图 5.3.40 所示，或与 SIL 模块的采样时间一致（0.01）。使用任何其他的采样时间都将导致 SIL 模块无法正确读取输入信号，进而无法得到正确的仿真结果。

Constant value:

1

☑ Interpret vector parameters as 1-D

Sampling mode: Sample based

Sample time:

0

图 5.3.40 模块采样时间

正确设置了输入模块的采样时间，再进行 SIL 测试，如图 5.3.41 所示，结果与图 5.3.28～图 5.3.30 所示一致。

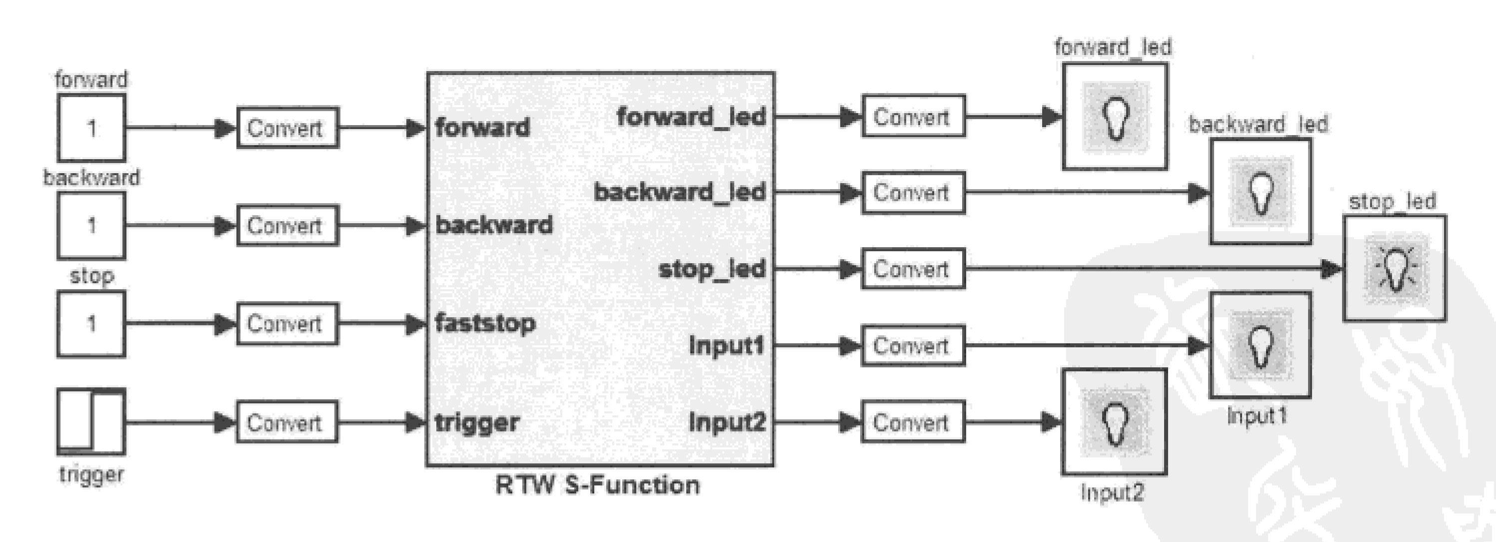


图 5.3.41 软件在环测试结果

5. 自动生成代码及编译

参考第 5.3.2 小节的“5. 自动生成代码及编译”内容。

6. 虚拟硬件测试

根据本章第 5.1 节的介绍，建立 proteus 电动机控制模型，如图 5.3.42 所示，并加载先前生成的 HEX 文件，单击仿真按钮，停转指示灯亮起，说明系统已启动。

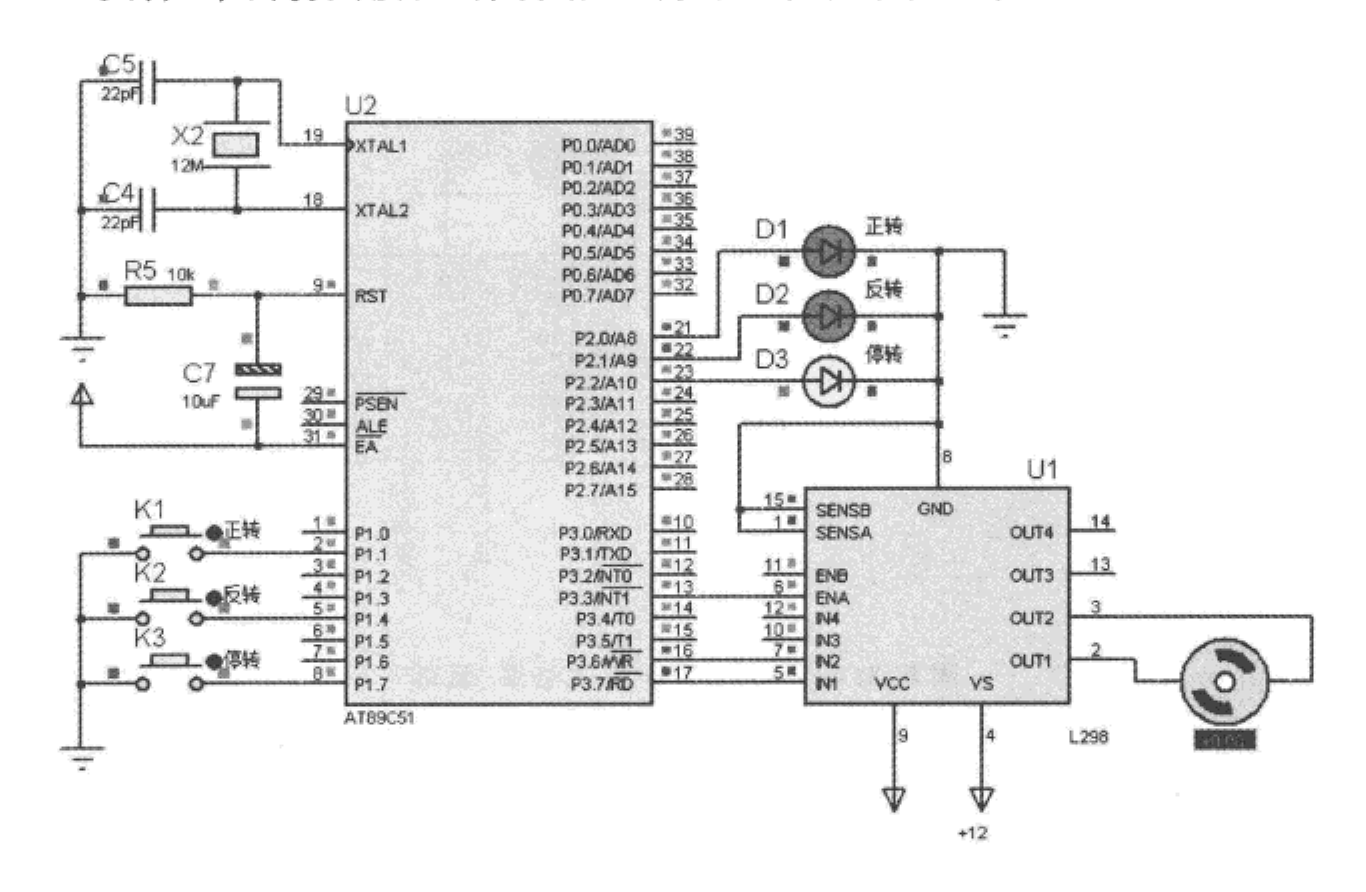


图 5.3.42　虚拟硬件测试结果：停转

分别按下正转、反转按钮，得到仿真结果如图 5.3.43、图 5.3.44 所示，说明正确实现了直流电动机的控制。

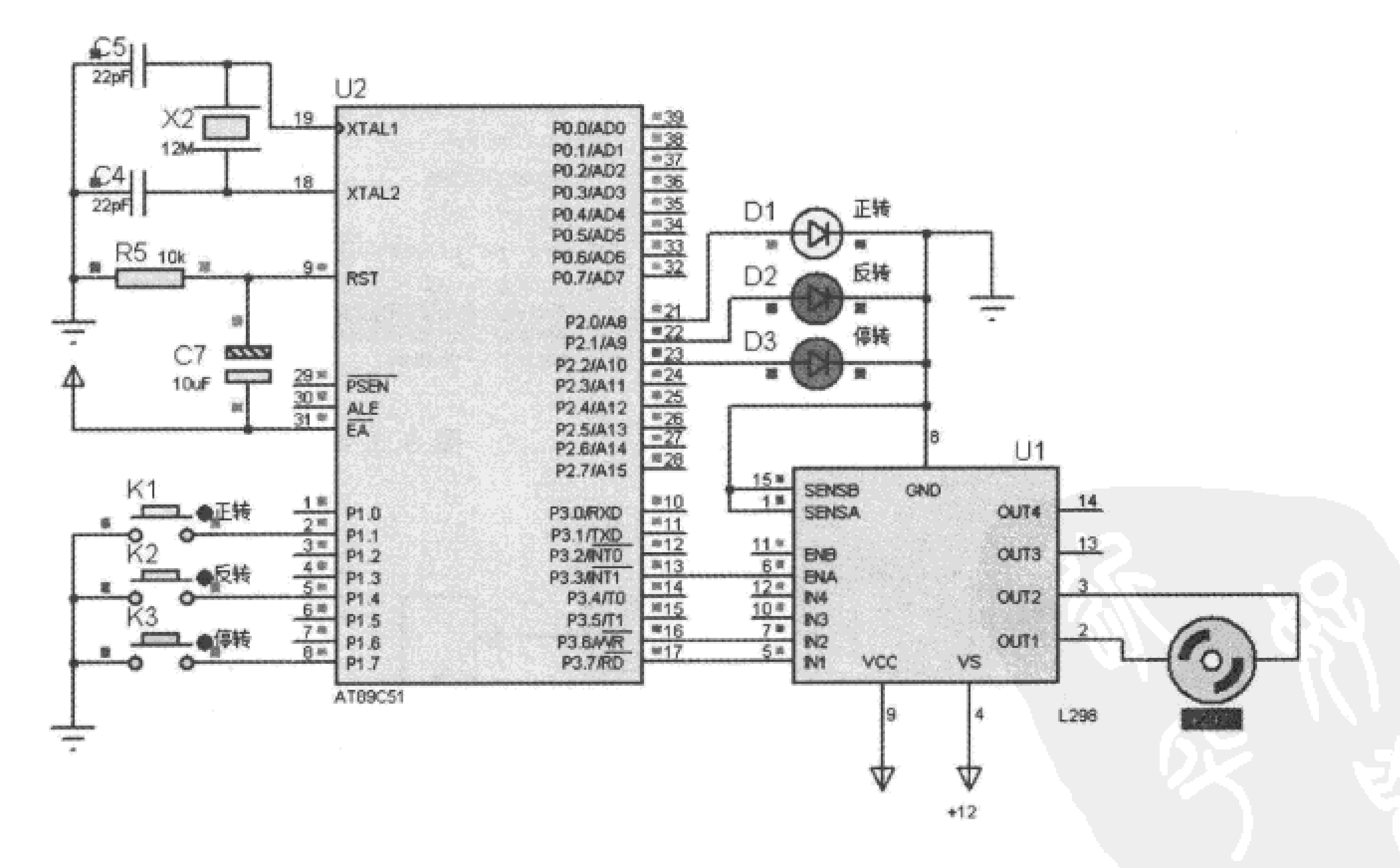


图 5.3.43　虚拟硬件测试结果：正转



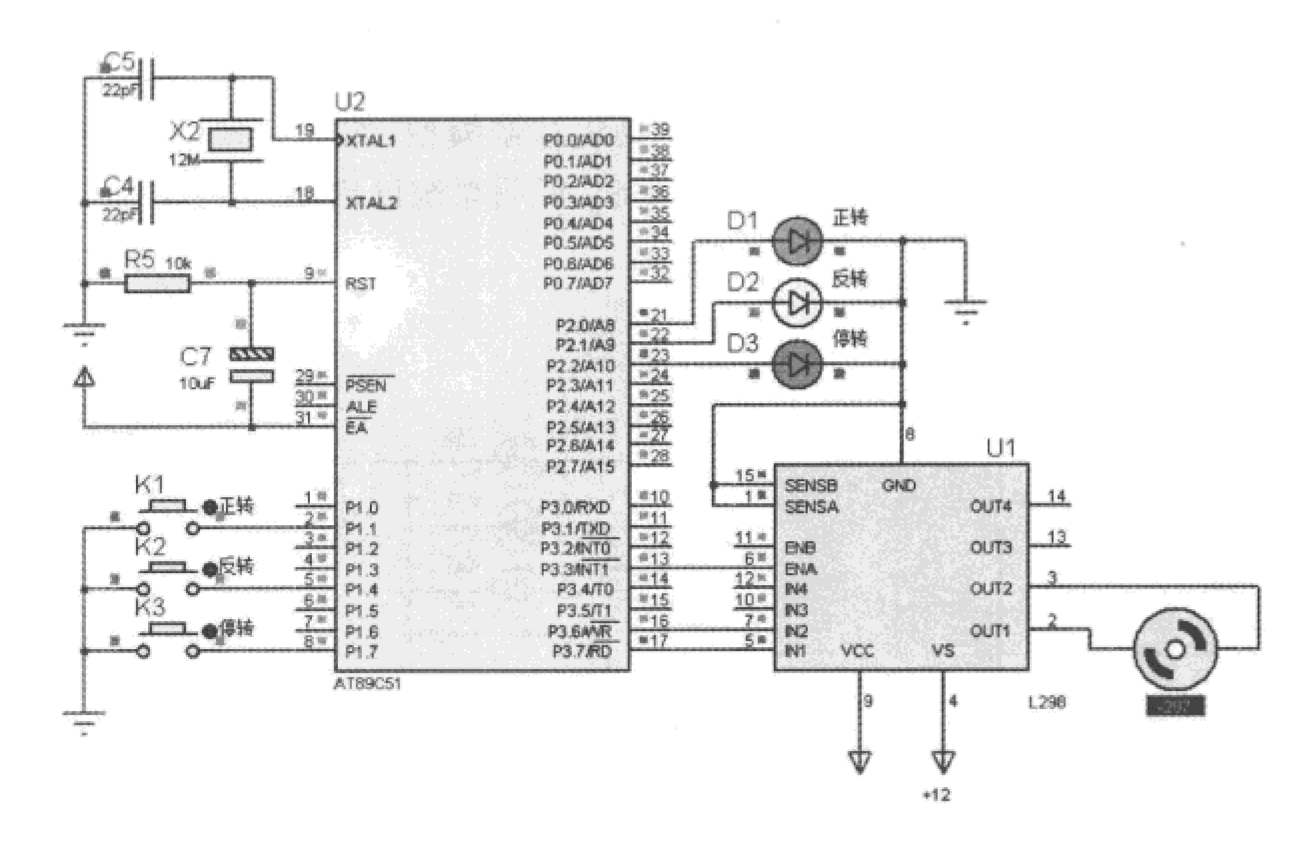


图 5.3.44　虚拟硬件测试结果:反转

5.3.3　算术乘法

1. 功能验证模型

在 Simulink 模块库中找到图 5.3.45～图 5.3.47 所示模块，并按图 5.3.48 连接，即可得到算术乘法模型。

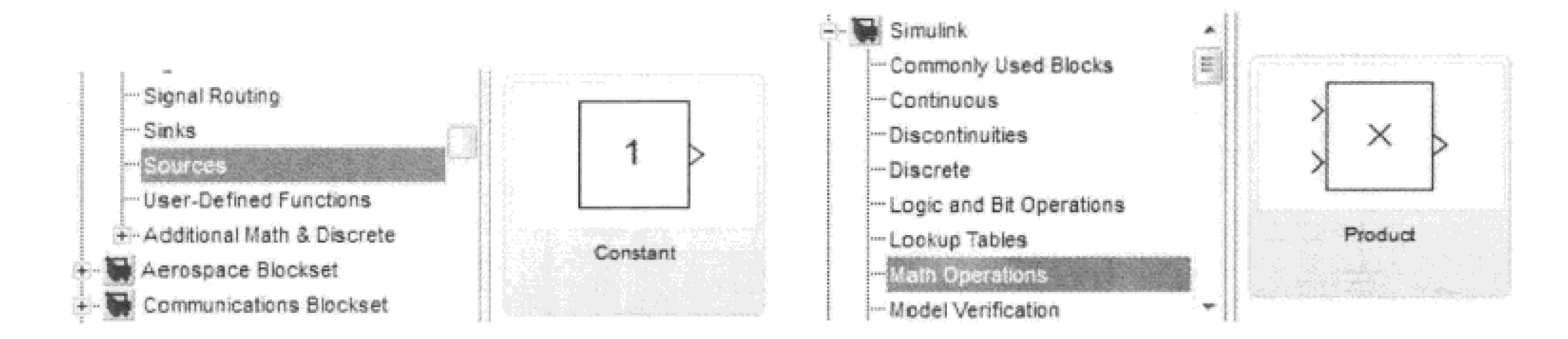


图 5.3.45　Constant 模块　　　　图 5.3.46　Product 模块

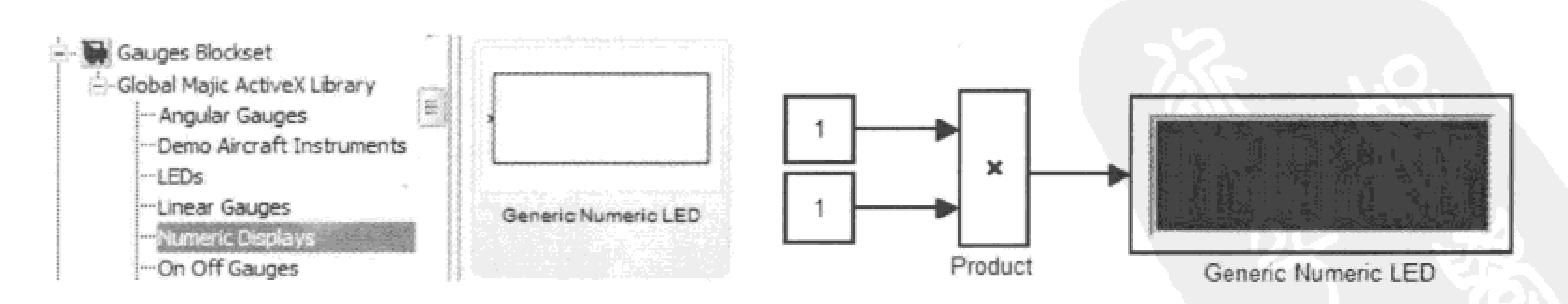
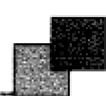


图 5.3.47　数字 LED 模块　　　　图 5.3.48　算术乘法模型

2 位十进制整数乘法的乘积最大为 4 位十进制整数，因此设置数码管的显示位数为 4 位整数、0 位小数，如图 5.3.49 所示。



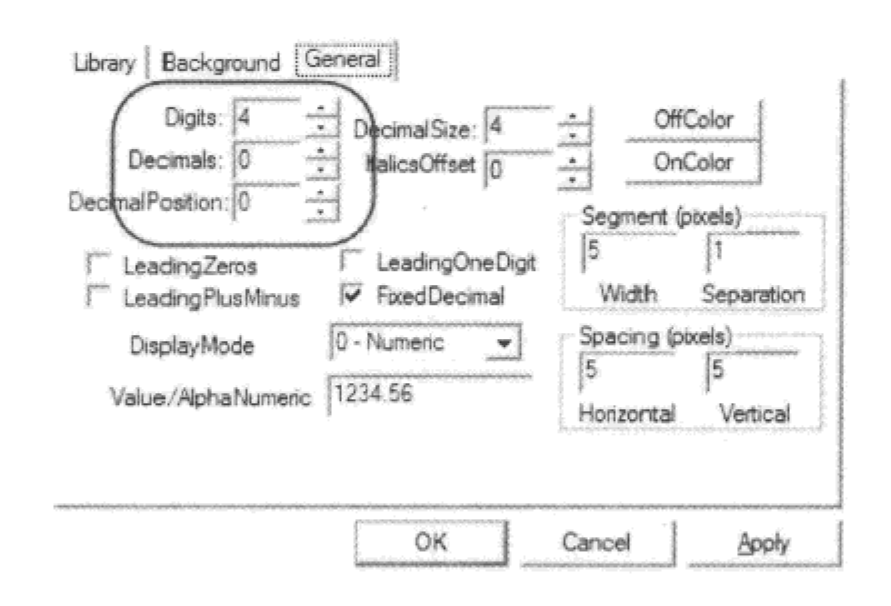


图 5.3.49　设置 LED 显示位数

选择模型主窗口的菜单项 Simulation→Configuration Parameters…，打开模型参数对话框，在 Solver 面板中，设置求解器为定步长离散求解器，步长为 0.01，如图 5.3.50 所示。

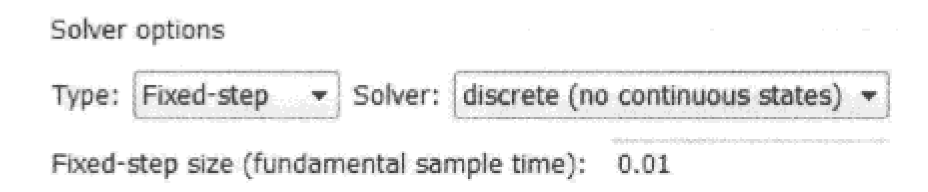


图 5.3.50　求解器设置

仿真结果如图 5.3.51、图 5.3.52 所示。

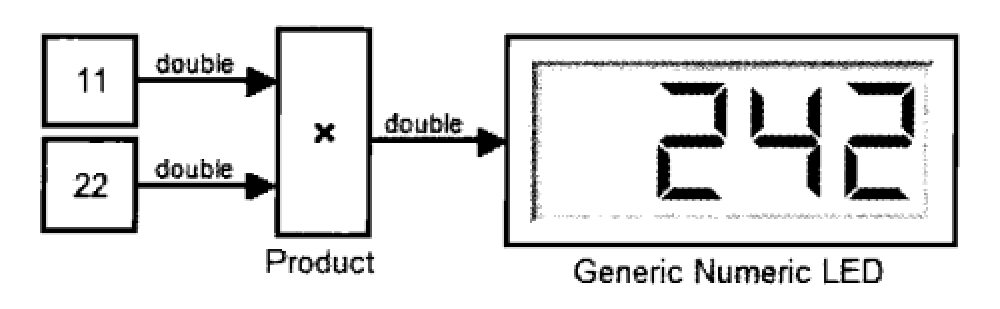


图 5.3.51　功能验证模型仿真结果 1

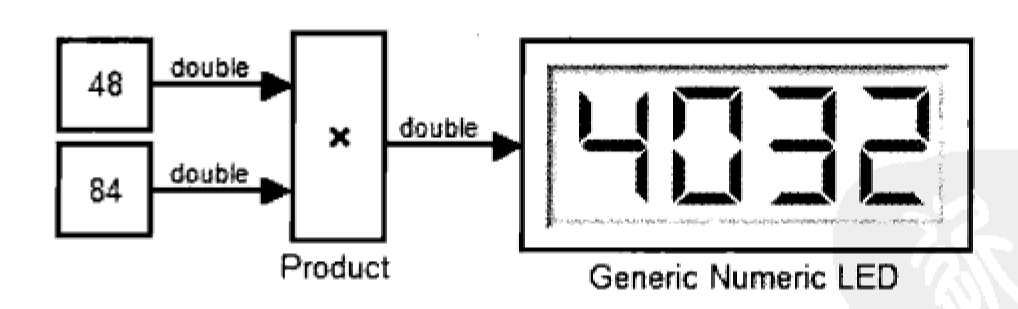


图 5.3.52　功能验证模型仿真结果 2

2. 软件在环测试

(1) 数据类型转换。在模块库 Simulink→Ports & Subsystems 中找到输入模块与输出模块，替换上述图 5.3.51 中的常数与 LED 模块，并将模型另存。

单击模型窗口的按钮，打开模型浏览器，将 Simulink 模型中的 In 模块的数据类型设置

为 uint8，Product 模块的数据类型设置为 uint16，Out 模块的数据类型可设为自动继承，也可强制设置为 uint16，如图 5.3.53 所示。

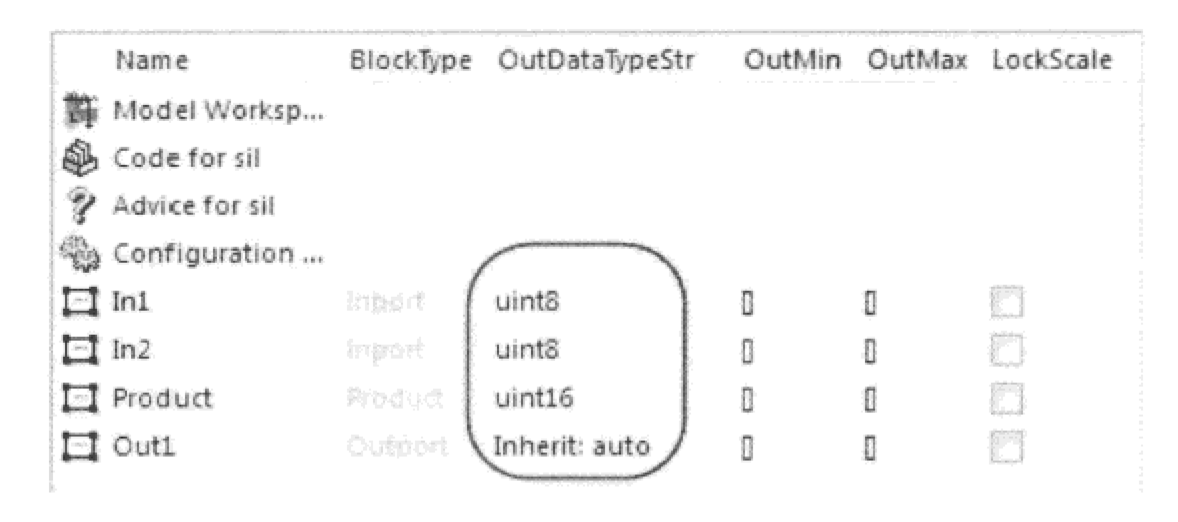


图 5.3.53　设置模型端口的数据类型

修改后的模型如图 5.3.54 所示。

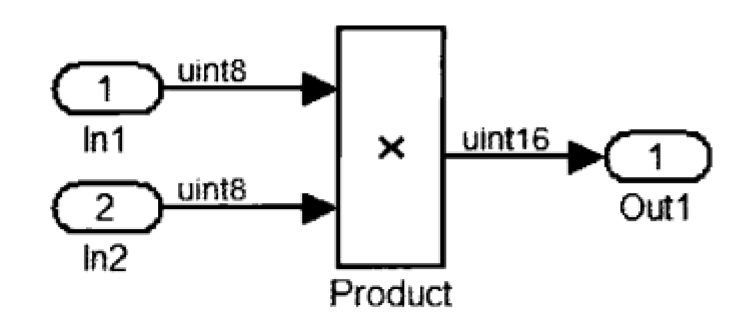


图 5.3.54　代码生成模型

（2）模型参数设置。打开模型参数对话框，在 Real-Time Workshop 面板设置 TLC 文件为 ert.tlc，如图 5.3.55 所示。

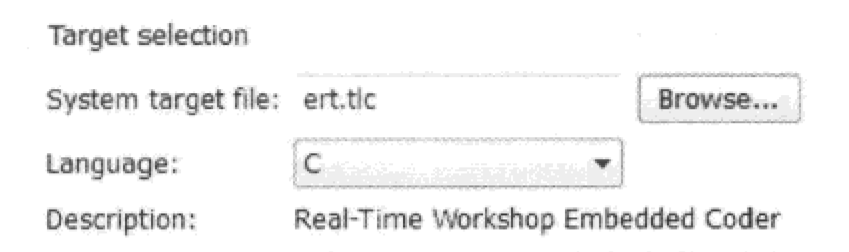


图 5.3.55　选择 TLC 文件

在 Real-Time Workshop→Interface 面板中，取消勾选不必要的复选框，如图 5.3.56 所示。

Software environment

Target function library: C89/C90 (ANSI)

Utility function generation: Auto

Support: floating-point numbers　non-finite numbers　complex numbers

absolute time　continuous time　non-inlined S-functions

variable-size signals

Multiword type definitions: System defined

图 5.3.56　Interface 面板

进入 Real-Time Workshop→Report 面板，勾选所有复选框，便于后期检查及跟踪，如图 5.3.57所示。

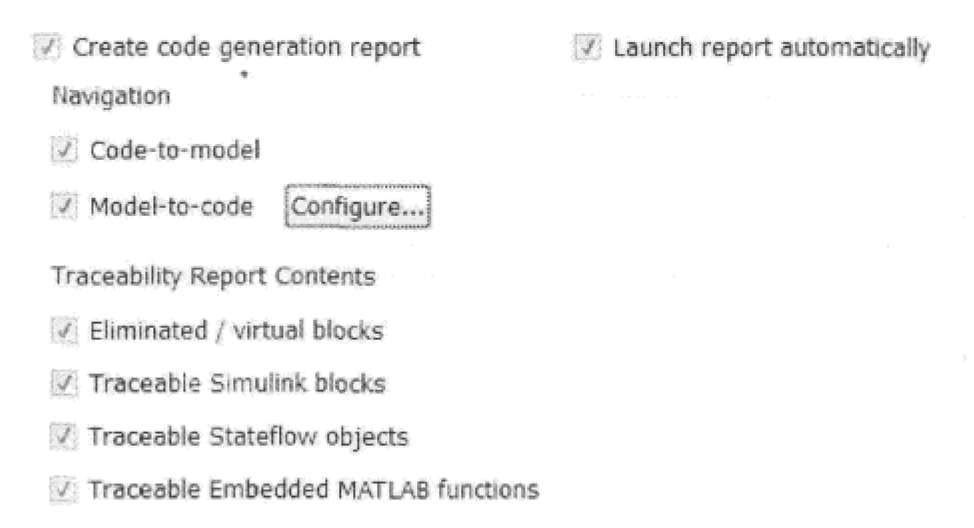


图 5.3.57　生成报告设置

(3) 生成 SIL 模块。在 Real-Time Workshop→SIL and PIL Verification 面板的 Create block 下拉列表，选择 SIL 选项，如图 5.3.58 所示。

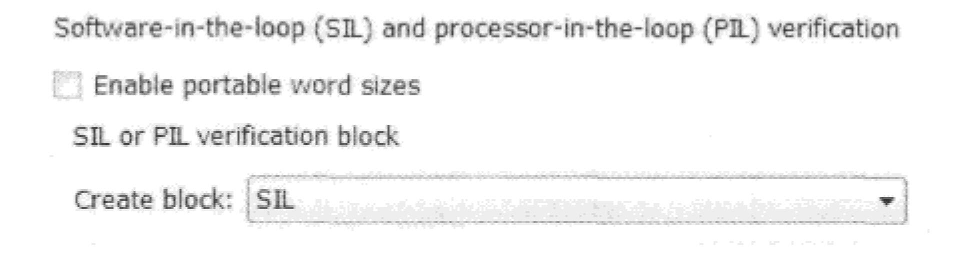


图 5.3.58　选择 SIL 选项

之后单击模型工具栏的按钮，得到代码生成报告与 SIL 模块，如图 5.3.59 和图 5.3.60 所示。

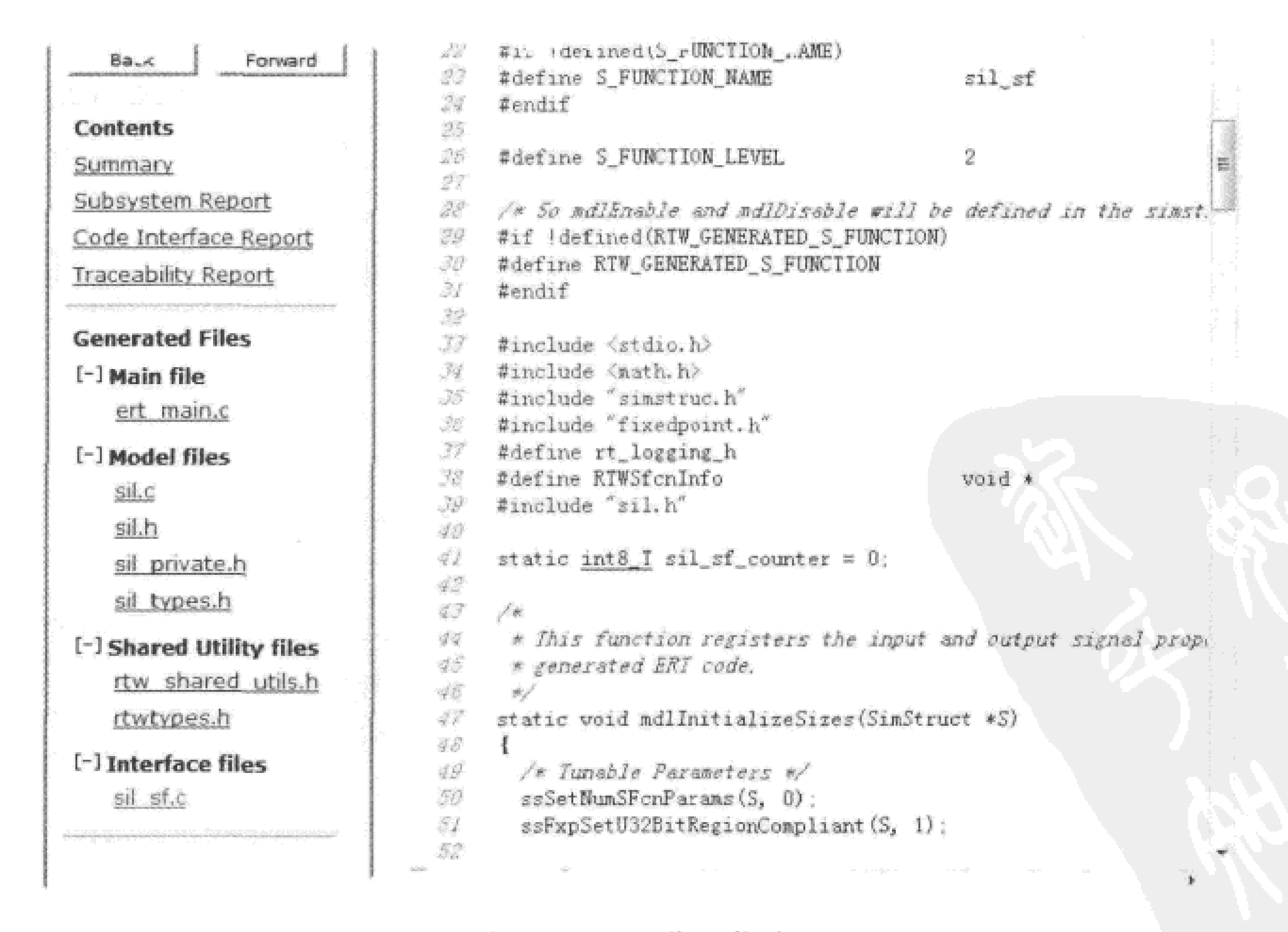


图 5.3.59　代码报告

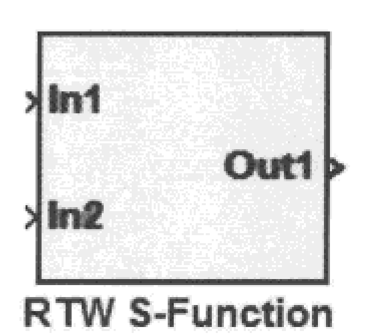


图 5.3.60　SIL 模块

以 SIL 模块替换图 5.3.51 的 Product 模块，重建验证模型，并在各端口间加入必要的数据类型转换模块。SIL 测试的结果图 5.3.61、图 5.3.62 与图 5.3.51、图 5.3.52 所示结果是一致的。

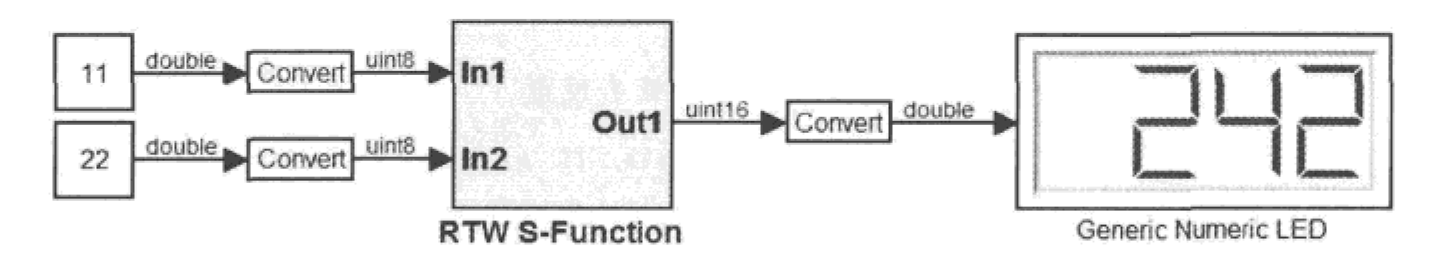


图 5.3.61　软件在环测试结果 1

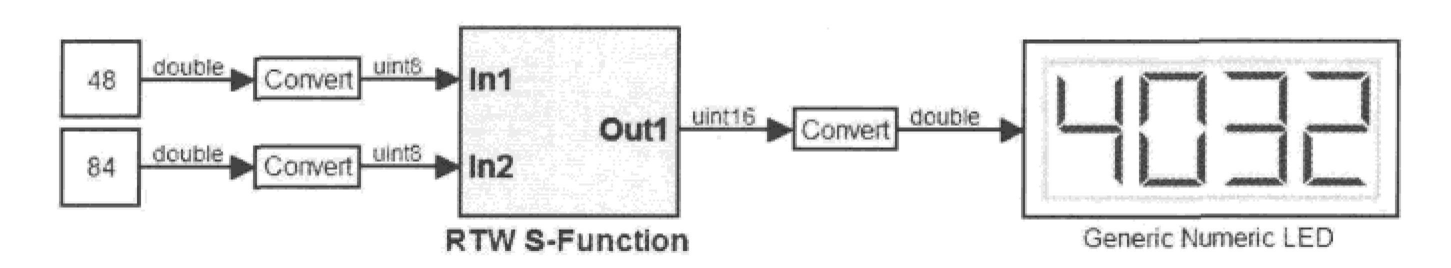


图 5.3.62　软件在环测试结果 2

3. 定义输入/输出信号

完成了软件在环测试，用户就可以参照 5.2.2 节、5.2.3 节或 5.3.2 节的说明，自动生成代码并在代码中添加必要的硬件接口代码，完成整个设计，不过本节采用另一种方式来完成：事先将硬件接口代码封装成一个函数，之后在生成的代码里直接调用。

选择 In1 端口连接线的右键菜单项 Signal Properties…，如图 5.3.63 所示，信号命名为 multiplicand，存储类型为 ImportedExtern，如图 5.3.64 所示。

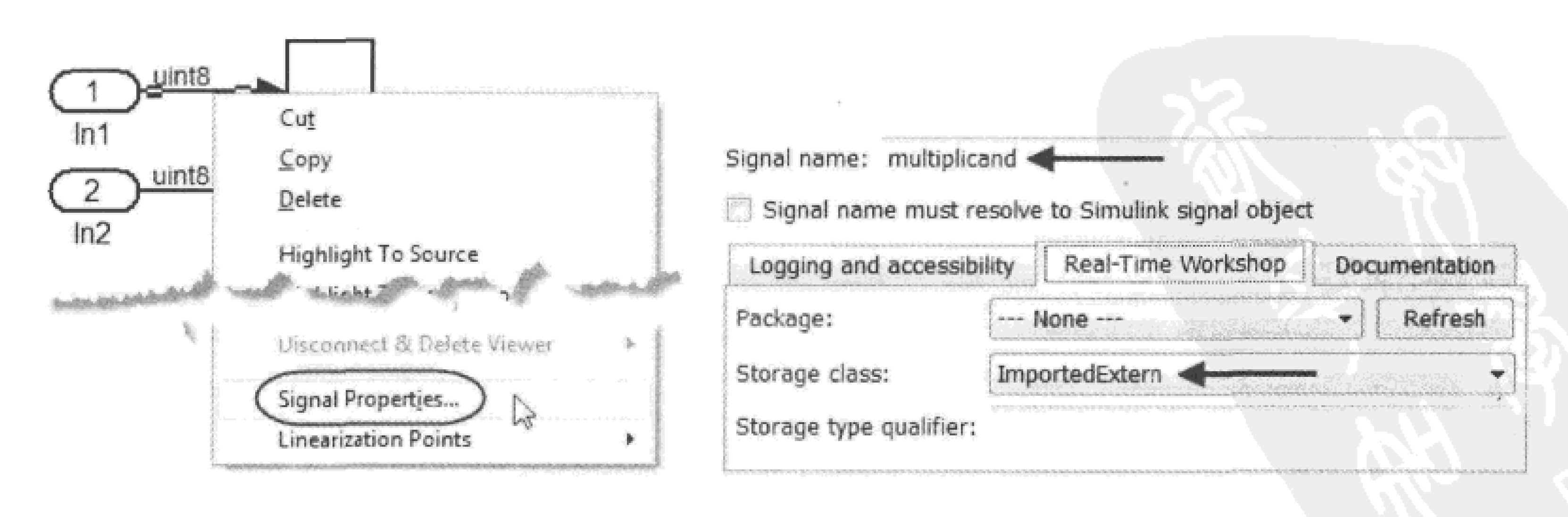
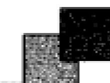


图 5.3.63　信号属性菜单项　　　　图 5.3.64　In1 端口信号设置



同样命名 In2 端口的信号为 multiplicator，存储类型为 ImportedExtern，如图 5.3.65 所示。使用 RTW 生成代码时，变量 multiplicand 与 multiplicator 将被声明为外部变量，因此用户需要在编写硬件接口代码时定义它们。

Out1 端口的信号命名为 product，存储类型为 ExportedGlobal，如图 5.3.66 所示。该变量将被定义在 RTW 生成的代码中，因此用户需要在编写硬件接口代码时将其声明为外部变量。

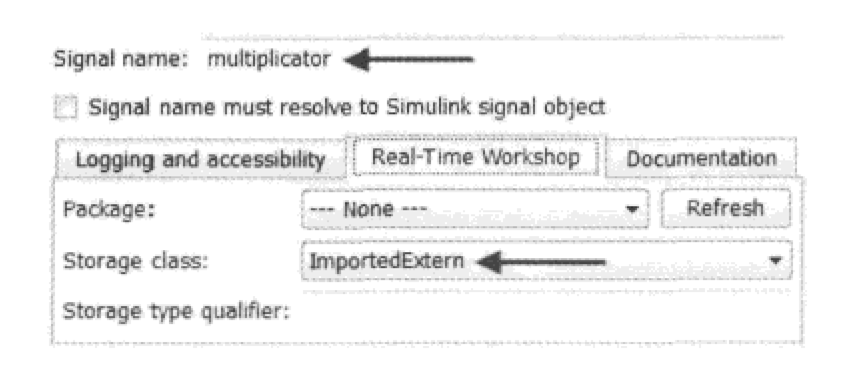


图 5.3.65　In2 端口信号设置

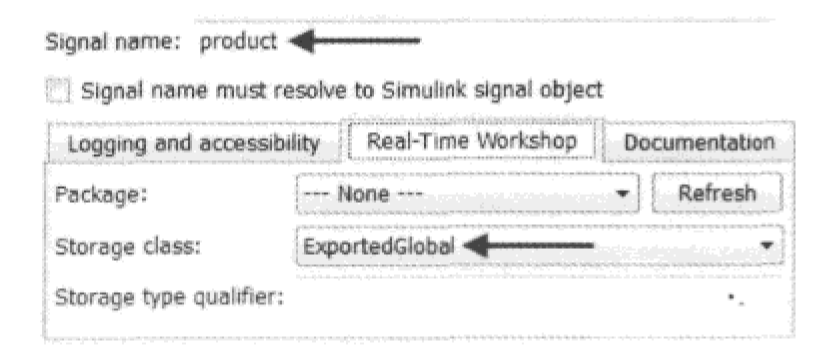


图 5.3.66　Out1 端口信号设置

打开模型菜单项 Format→Port/Signal Displays，选择 Port Data Types 与 Signal Class 命令，如图 5.3.67 所示，各条信号线即可显示数据类型、数据存储类型以及信号名，如图 5.3.68 所示。

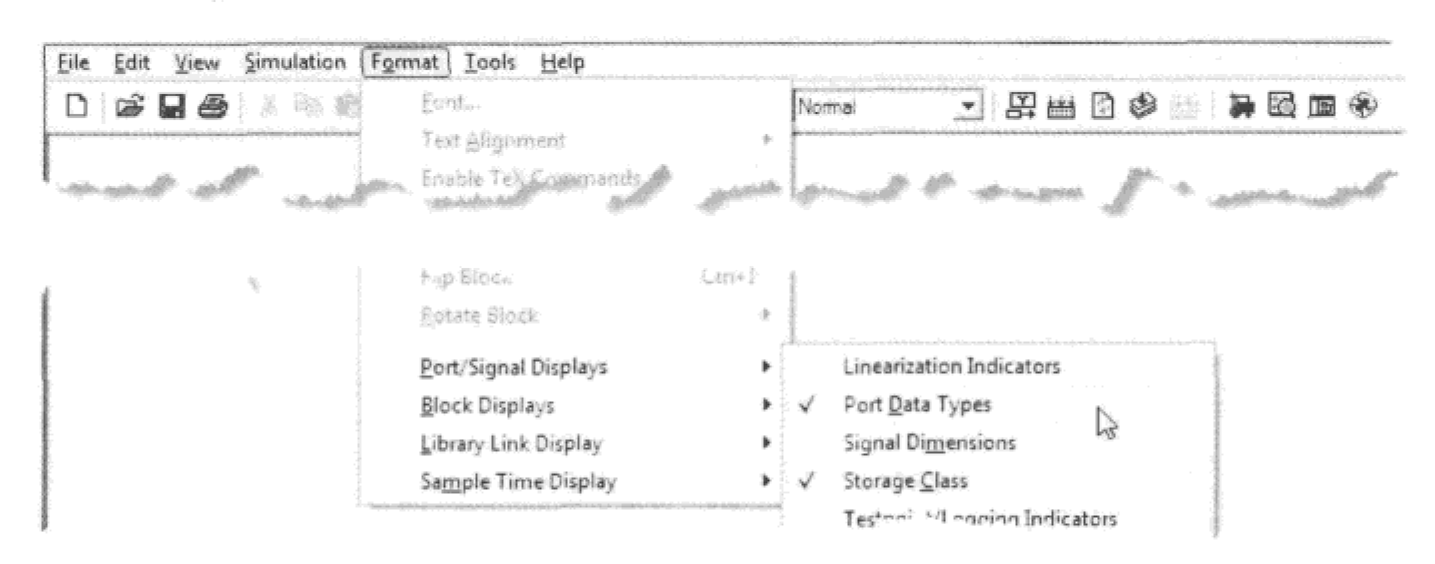


图 5.3.67　开启端口数据类型与存储类型显示

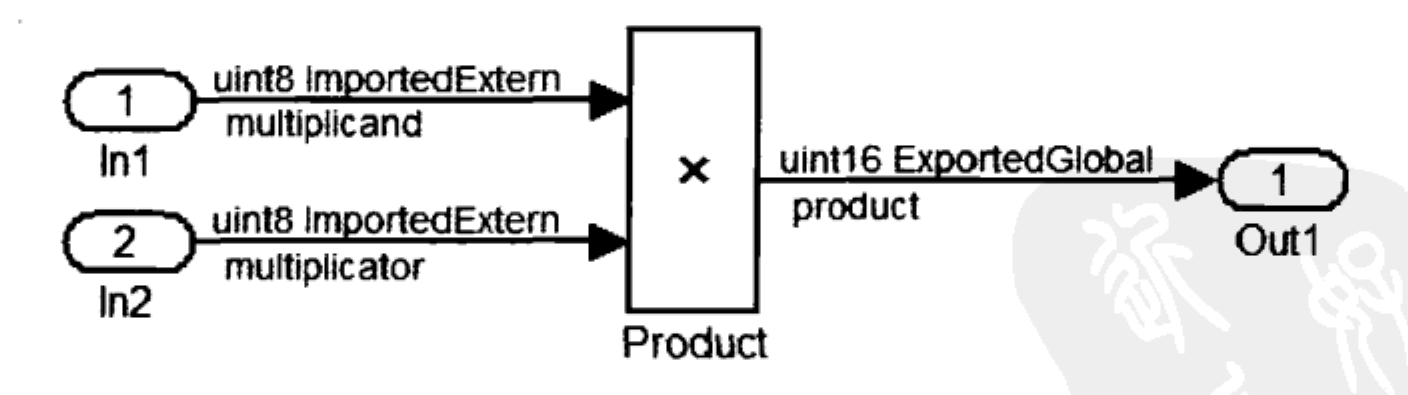


图 5.3.68　各条信号线上的信号属性

4. 硬件接口代码

根据上文定义的三个变量 multiplicand、multiplicator、product 的存储类型，在 io.h 文件中声明这三个变量。

```
#include "rtwtypes.h"                // RTW 数据类型头文件
uint8_T multiplicand;                // multiplicand 声明为本地变量
uint8_T multiplicator;               // multiplicator 声明为本地变量
extern uint16_T product;             // product 声明为外部变量
void input(void);                    // 接口函数声明
void output( void );                 // 接口函数声明
```

在 io.c 文件中，编写输入/输出的硬件接口代码。

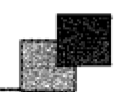
```
#include "io.h"
 void input()
{   int multiplicand1;              // 中间变量
    int multiplicand2;
    int multiplicator1;
    int multiplicator2;
    multiplicand1 = P1 & 0x0F;                  // 取输入的低 4 位
    multiplicand2 = (P1 & 0xF0)>> 4;            // 取输入的高 4 位
    multiplicand = multiplicand2 * 10 + multiplicand1;          // 组合成被乘数
    multiplicator1 = P0 & 0x0F;                 // 取输入的低 4 位
    multiplicator2 = (P0 & 0xF0)>> 4;           // 取输入的高 4 位
    multiplicator = multiplicator2 * 10 + multiplicator1;       // 组合成乘数
} void output()
{   int product1;              // 中间变量
    int product2;
    int product3;
    int product4;
    product4 = product/1000;            // 取乘积的千位
    product3 = product/100 % 10;        // 取乘积的百位
    product2 = product/10 % 10;         // 取乘积的十位
    product1 = product % 10;            // 取乘积的个位
    P2 = product2 << 4|product1;        // 千位、百位输出
    P3 = product4 << 4|product3;        // 十位、个位输出
}
```

5. 自动生成代码及编译

参考第 5.3.2 小节的“5. 自动生成代码及编译”内容。

6. 虚拟硬件测试

根据本章第 5.1 节的介绍，建立 proteus 算数乘法模型，并加载先前生成的 HEX 文件，单击仿真按钮，任意设置被乘数与乘数，得到仿真结果如图 5.3.69、图 5.3.70 所示，说明正确实现了 2 位十进制算术乘法。



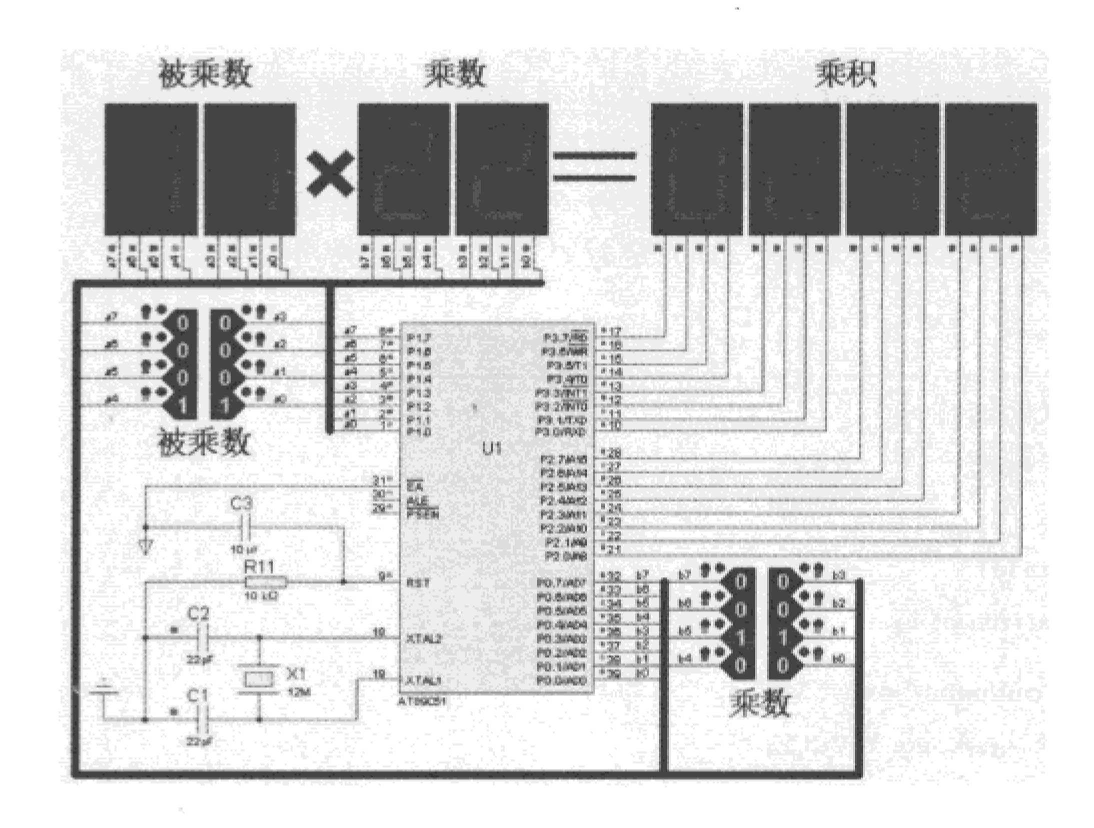


图 5.3.69　虚拟硬件测试结果 1

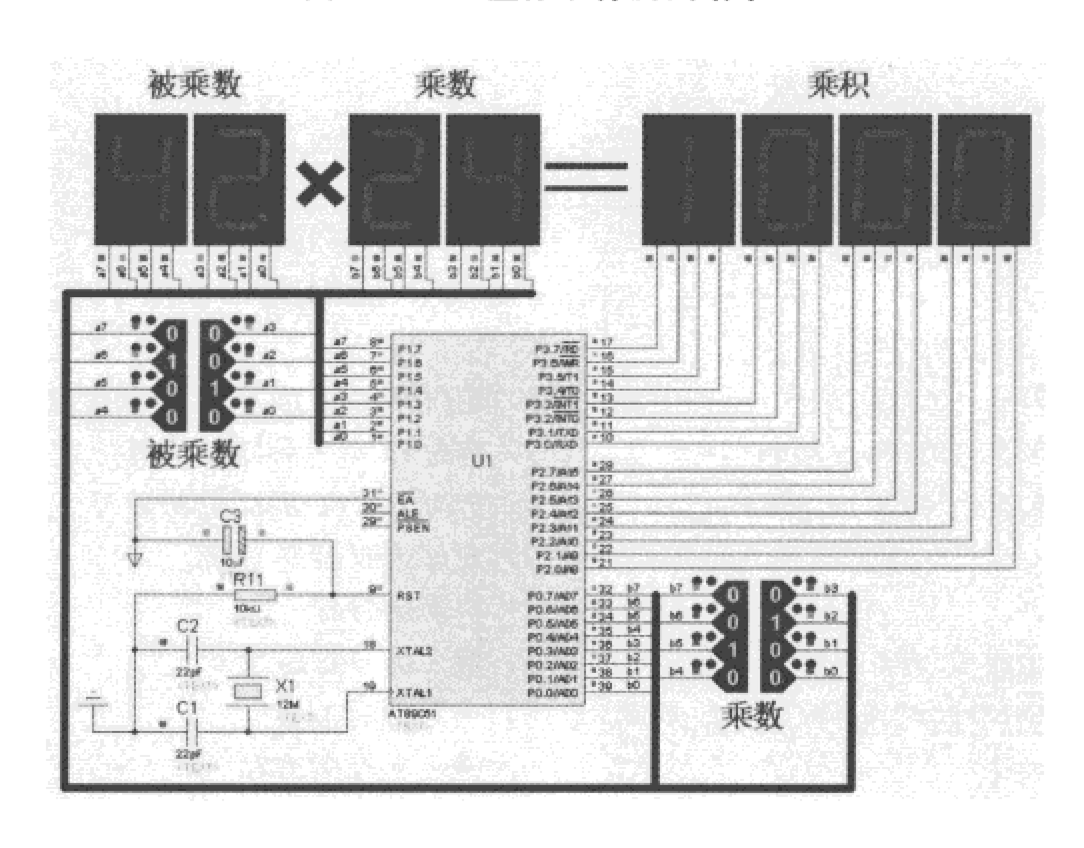


图 5.3.70　虚拟硬件测试结果 2

5.3.4　流水灯

本节重用5.2.3节所建立的流水灯模型，使用TASKING EDE生成HEX文件。读者看到这里应已了解，整个设计过程与前文的差别仅在于修改硬件接口代码，因此这里列出需要调整的代码段，其他步骤皆可参照5.2.3节与5.3.2节。

```
...
#include <stdio.h>                  //删除该头文件
#include "light_tasking.h"
#include "rtwtypes.h"
```

```
int num = 0;                              // 定时标志
_interrupt(1) void tic( void )            // TASKING EDE 的中断程序定义方式
{   num + + ;                             // 定时标志
    TH0  = (65535 - 50000)/256;           // 定时 0.05s 的高位初值
    TL0  = (65535 - 50000)%256;           // 定时 0.05s 的低位初值
}
```

```
void rt_OneStep(void);
void rt_OneStep(void)
{   ...
    // Step the model
    light_tasking_step();
    // Get model outputs here
    P2_0  =  light_tasking_Y.Out1;
    ...
    P2_7  =  light_tasking_Y.Out8;
   //将输出 light_tasking_Y.Out1 等与实际硬件端口连接
    ...
}
```

```
int_T main();                                   //删除不必要的输入参数
int_T main()                                    //删除不必要的输入参数
{
     TMOD  =  0x01;                             // 定时器 0 工作于16 位计时状态
     TH0  =  (65535 - 50000)/256;               // 定时 0.05s 的高位初值
     TL0  =  (65535 - 50000)%256;               // 定时 0.05s 的低位初值
     IE  =  0x82;                               // 允许定时器 0 中断
     TR0  =  1;                                 // 启动定时器
     light_tasking_U.In1  =  0;                 // 输入信号初始为 0
     while (rtmGetErrorStatus(light_tasking_M) == (NULL))
     {   if(num ==1)                            // 满足定时标志时
         {   num = 0;                           // 定时标志清零
             light_tasking_U.In1  =  ~light_tasking_U.In1;
         }                                      // 翻转输入信号,实现 Stateflow 脉冲信号
         rt_OneStep();
     }
...
}
```

编译结果如图 5.3.71 所示。

在 Proteus 模型中加载该 HEX 文件,可以得到图 5.2.69 所示的仿真结果(图 5.3.72)。

用户可以比较使用 Keil 与 TASKING EDE 两款编译环境生成的 HEX 文件,如图 5.3.73 所示,由 Keil 编译得到的 HEX 文件有 2807 字节,而 TASKING EDE 编译得到的 HEX 文件有 1874 字节,显然后者的编译效率高于前者,这也就是本书介绍 TASKING EDE 编译环境的原因。



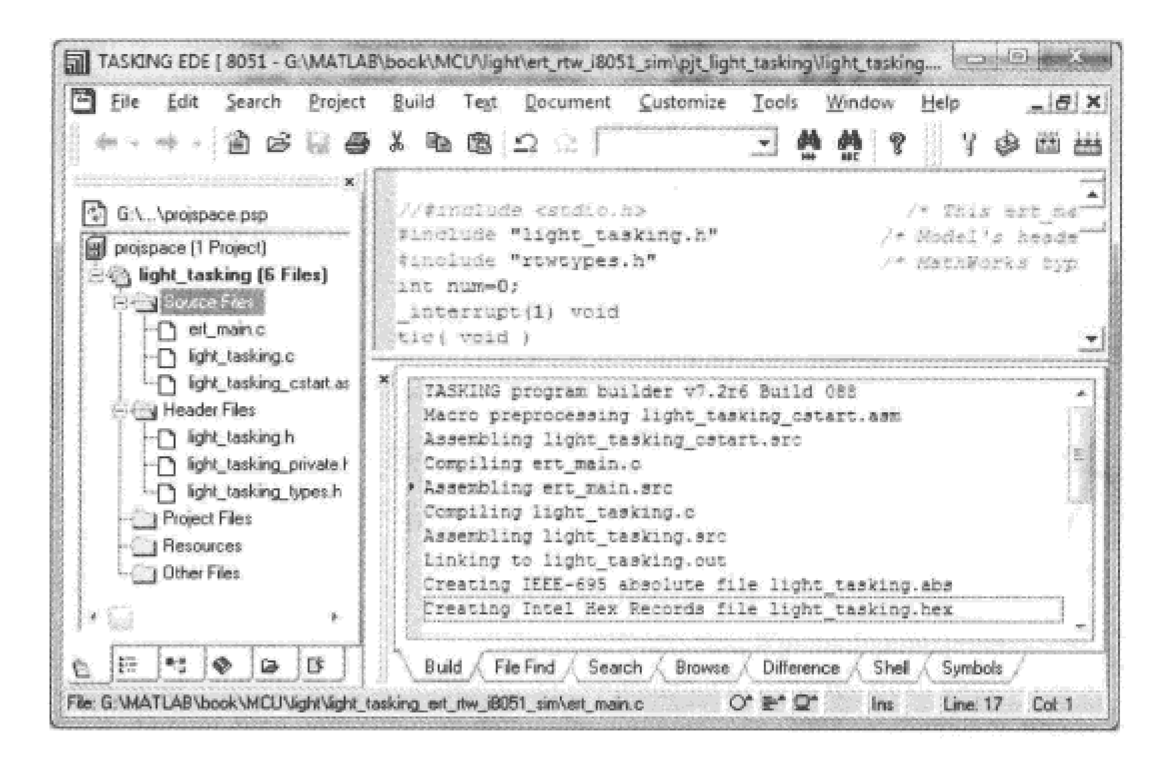


图 5.3.71　编译生成 HEX 文件

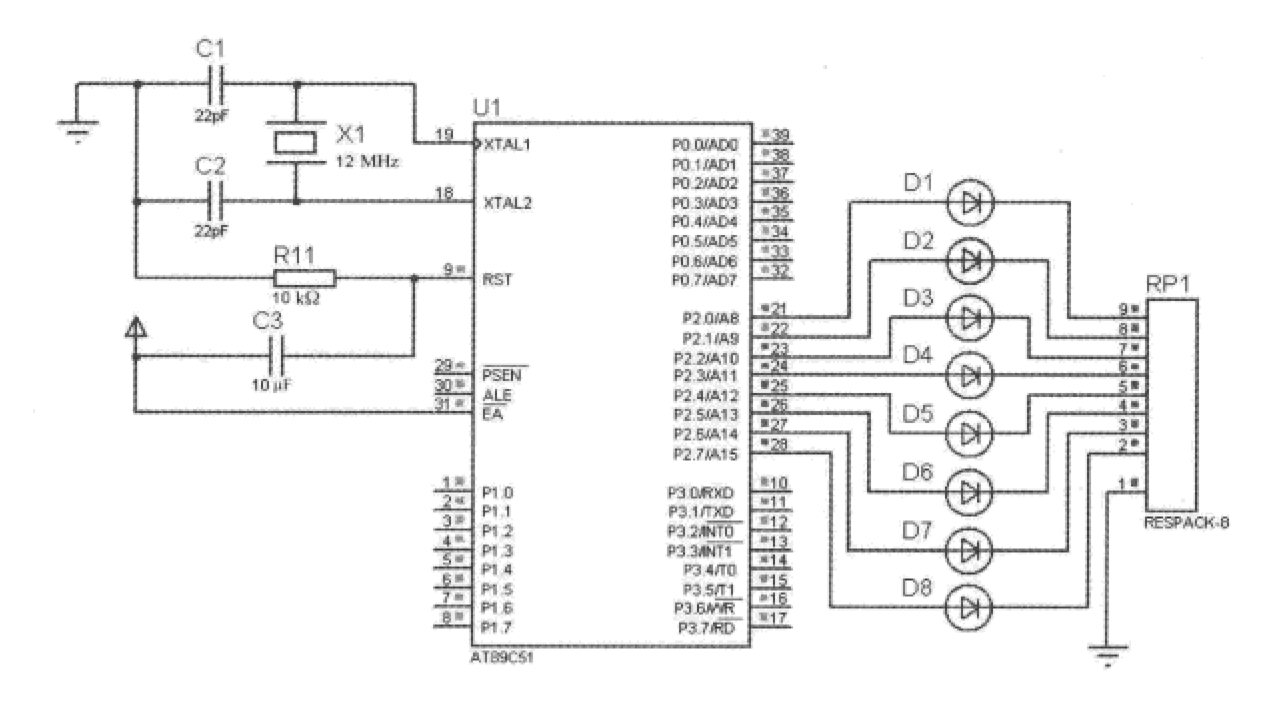


图 5.3.72　虚拟硬件测试结果

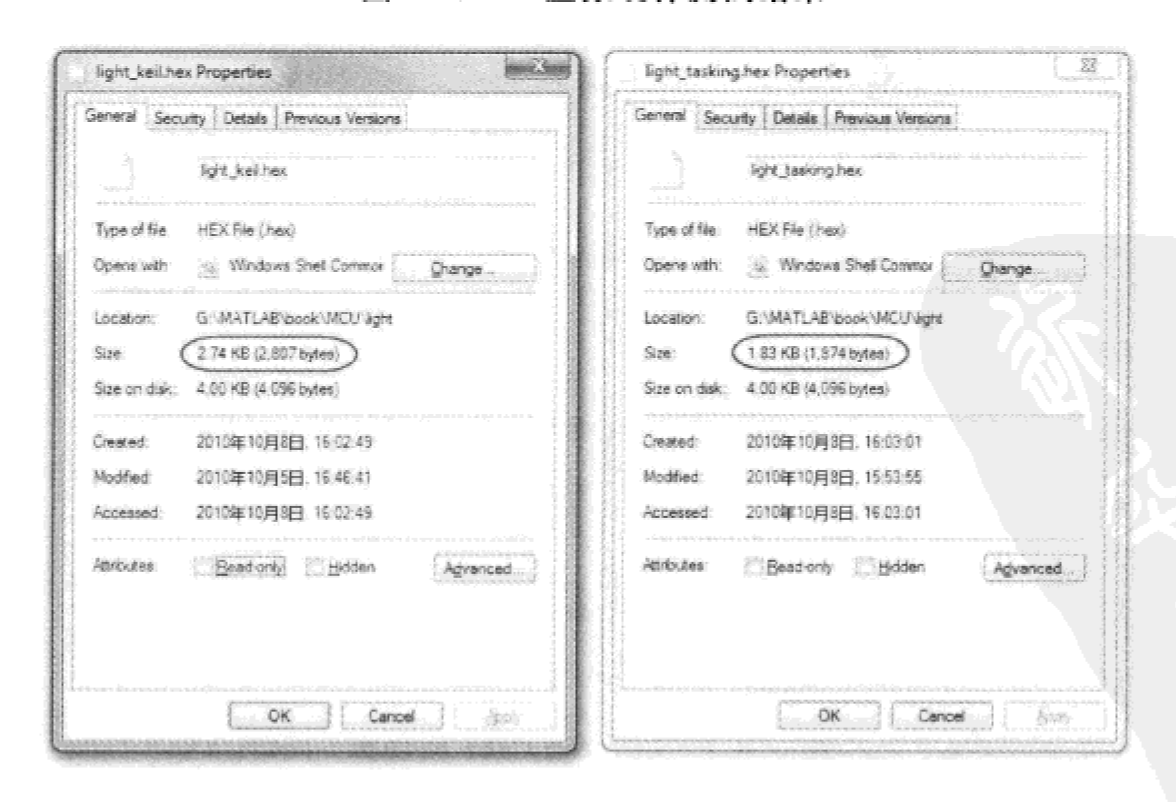


图 5.3.73　比较生成的 HEX 文件

第 6 章

C166 代码的快速生成

本章的主要目的是介绍 C166 单片机代码的自动生成和处理器在环测试(PIL)。处理器在环测试是将模型自动生成的嵌入式 C 代码下载到实际的处理器中,并和被控对象模型在模型中进行非实时性联合仿真,通过真实的 I/O 口、串口等来交换工作在处理器上的嵌入式 C 代码和运行在模型中被控对象模型间的数据,来评估算法的优劣。一般进行这样的测试是需要实际硬件的,不过 MathWorks 公司提供了在 TASKING EDE 中进行 PIL 测试的功能。但只支持 8051、飞思卡尔 DSP563xx 和英飞凌等单片机的处理器在环测试,不支持 ARM 处理器这种测试。第 5 章的 PIL 测试被换成了 SIL 测试,是因为作者的 TASKING EDE for 8051 软件是 Demo 版(功能受限),无法完成这种测试。通过本章的介绍,读者同样可以了解 8051 PIL 测试的流程。

英飞凌公司作为全球半导体创新的领导者,共拥有 41 000 项专利,英飞凌产品在全球得到了广泛的应用。特别是在汽车尾气排放的控制方面有自己独到的解决方案。MathWorks 公司对英飞凌单片机也比较感兴趣,如 C166 等处理器,很多外设都被写成了模块,放到了 Simulink 的模型浏览器中,如图 6.0.1 所示。

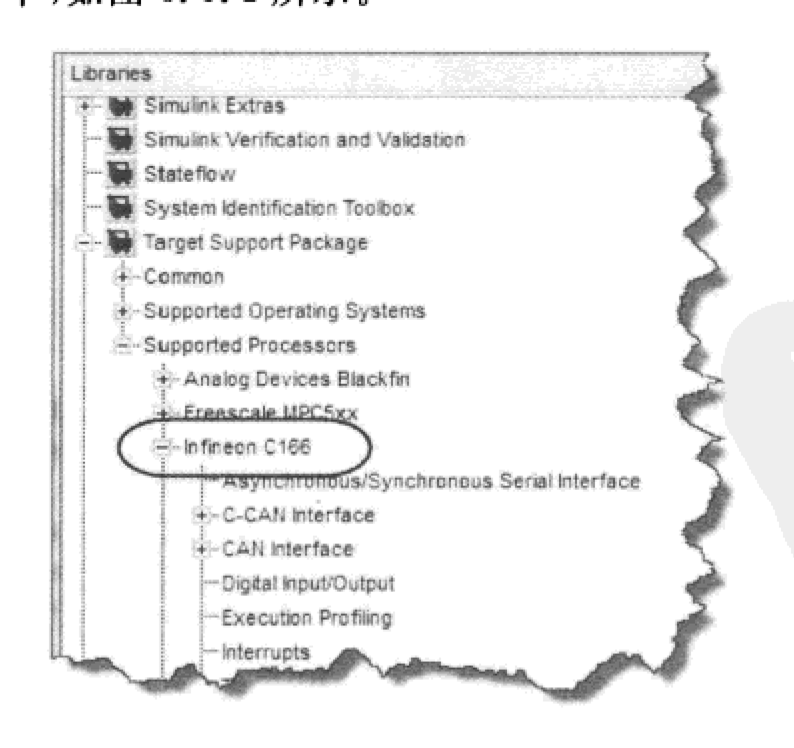
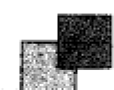


图 6.0.1　Infineon C166 模块库



为了满足习惯使用英飞凌产品这部分用户，这部分能利用基于模型设计的思想从事C166单片机的快速应用研发，作者将其单独作为一章来给读者作一简单介绍。由于作者没有C166的开发板，Proteus虚拟硬件实验平台又不支持英飞凌单片机，无法对自动生成的代码进行硬件测试。本章仅以第5章的电动机控制模型为例，来介绍C166代码的快速生成过程，以及进行功能验证与代码验证。不过，读者只要认真阅读了本章和其前后几章的内容，这种新技术是完全可用于英飞凌处理器开发的。

6.1 英飞凌C166模块库简介

Target Support Package是Mathworks公司提供的一款用于开发微处理器、微控制器、DSP等嵌入式系统的组件。它可以将芯片外设、实时操作系统以及由Simulink模型、Stateflow图表、Embedded MATLAB所生成的代码整合起来，避免了编写大量的底层驱动程序，可用于嵌入式系统的快速原型设计及实时性能分析等。

C166模块库是Traget Support Package的一个子集，与C166芯片的外围驱动设备一一对应，用户可以在算法模型中添加这些模块，用于生成可直接在嵌入式处理器上使用的代码。

在Simulink库浏览器中，打开Target Support Package→Support Prossessors→Infinion C166，即可打开C166模块库。其中包含有异步/同步串行接口、数字信号输入/输出、中断等子模块库，如图6.1.1所示。下面介绍C166中的部分子库。

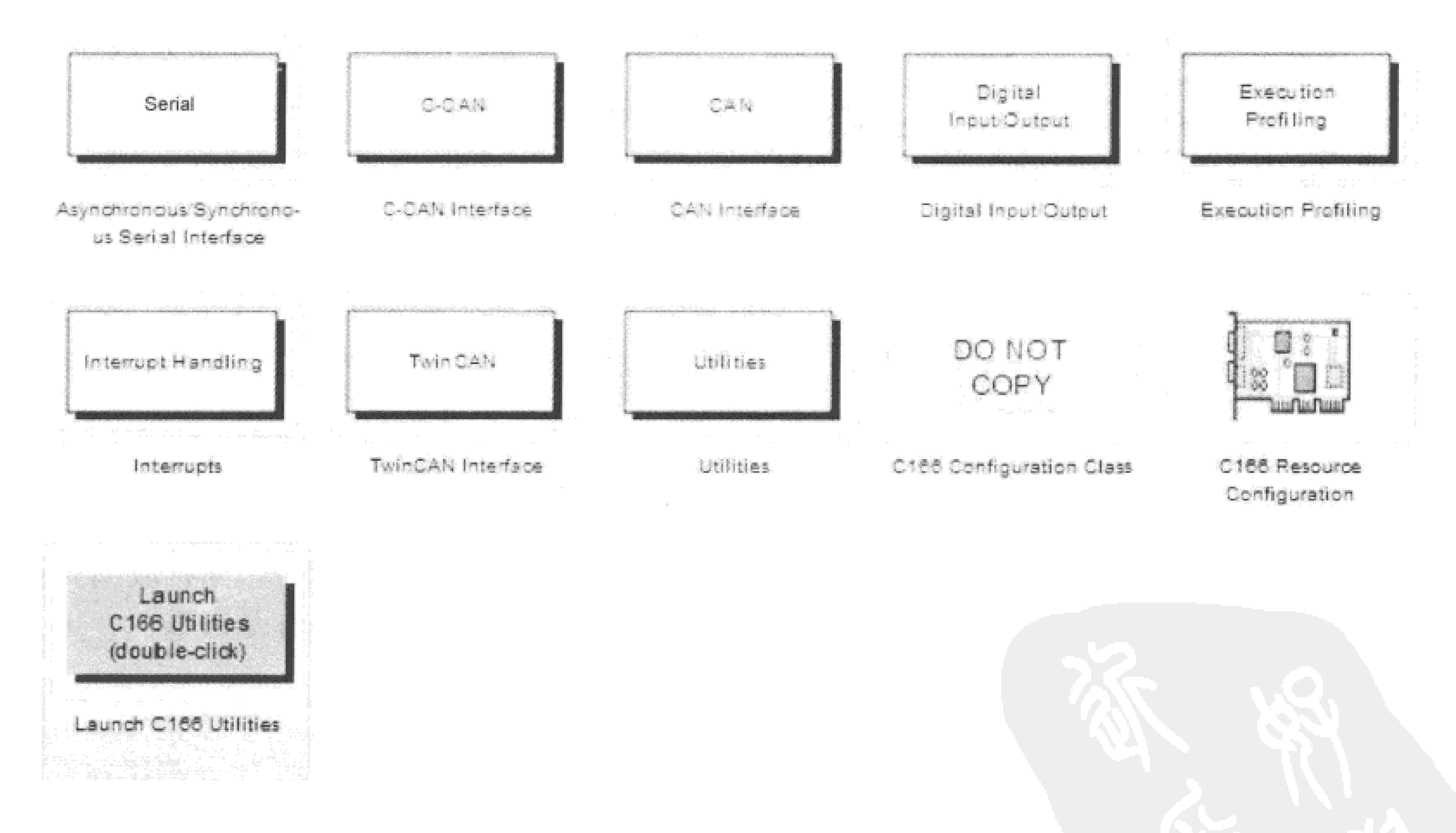


图6.1.1　Infinion C166模块库

(1) 异步/同步串行接口。该子库包含Serial Transmit 和Serial Receive两个模块，如图6.1.2所示。Serial Receive模块用于配置C166微控制器的串行接收功能，Serial Transmit 模块用于配置器串行发送功能。串行接收/发送模块通过Infineon C166微控制器的异步/同步串行接口ASC0通信。既可进行定字长通信，也可进行可变字长通信。

(2) 数字信号输入/输出。该子库包含 Digital In 和 Digital Out 两个模块，如图 6.1.3 所示。Digital In 和 Digital Out 模块用于从指定的端口或引脚输入/输出逻辑电平值。

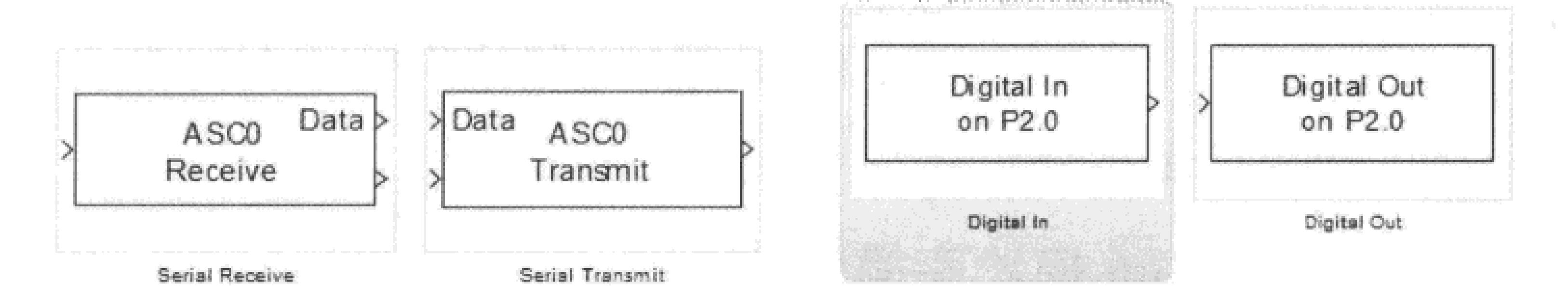


图 6.1.2　异步/同步串行接口模块　　　　图 6.1.3　数字信号输入/输出子库

(3) 中断。该子库包含模块 Fast External Interrupt，如图 6.1.4 所示。该模块在中断发生时触发一个异步函数调用。

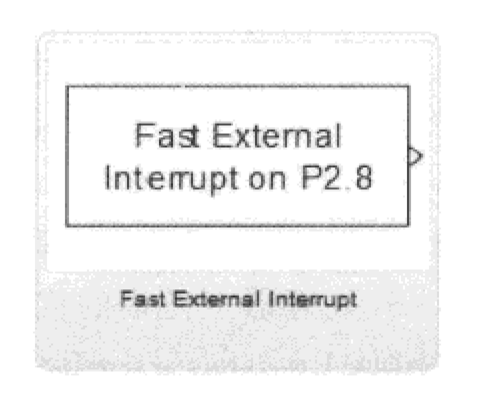


图 6.1.4　中断子库

函数调用子系统作为一个异步任务执行。此模块可为任务分配 Simulink 优先级和 CPU 中断等级。

(4) 执行分析。该子库包含模块 C166 Execution Profiling via ASC0/C-CAN 1/CAN A/TwinCAN A，如图 6.1.5 所示。

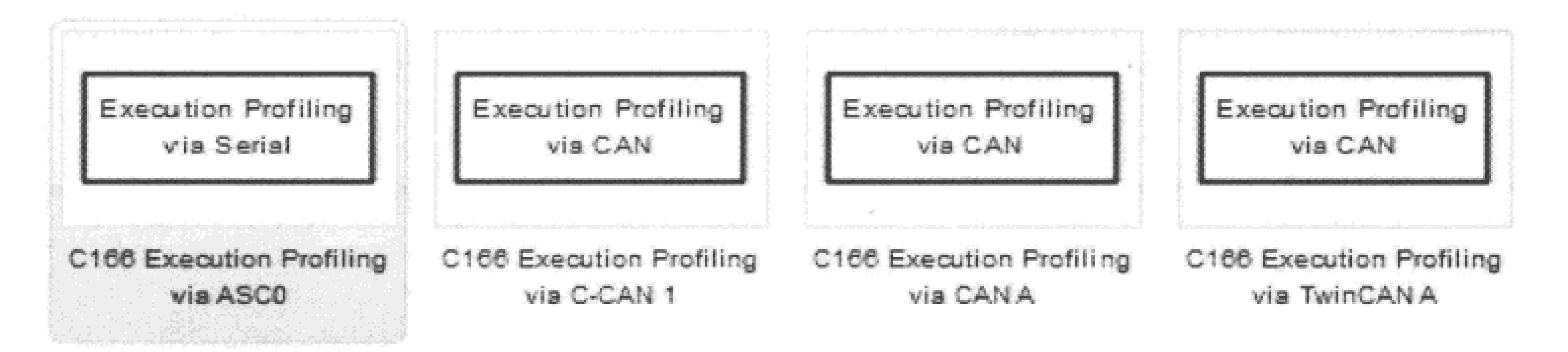


图 6.1.5　执行分析子库

C166 Execution Profiling via ASC0 为执行分析引擎提供串行接口，当接收到开始指令时，开始记录执行分析数据；当完成记录后记录下的数据自动通过串行接口 ASC0 返回。

C166 Execution Profiling via CAN A 为执行分析引擎提供了 CAN 接口，当接收到开始指令时，开始记录执行分析数据；当完成记录后记录下的数据自动通过 CAN 接口返回。用户需要为开始指令和返回数据指定标志信息。

C166 Execution Profiling via C-CAN 1 提供了 C-CAN 接口，其他功能与 C166 Execution Profiling via CAN A 相同。

(5) 工具。该子库包含模块 Switch External Mode Configuration 和 Switch Target Configuration，如图 6.1.6 所示。

Switch External Mode Configuration 将模型设置为外部模式或执行编译，在模型中添加该模块后，双击可打开其对话框，如图 6.1.7 所示。



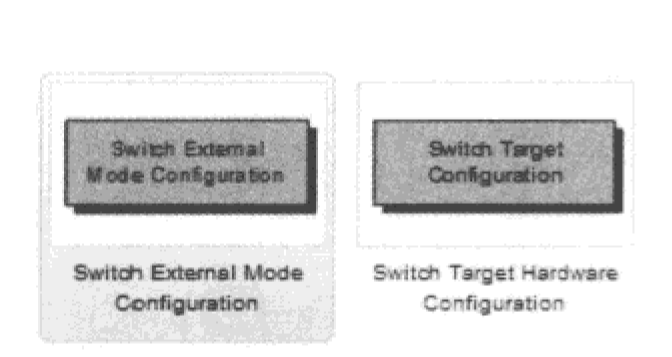


图 6.1.6　工具子库

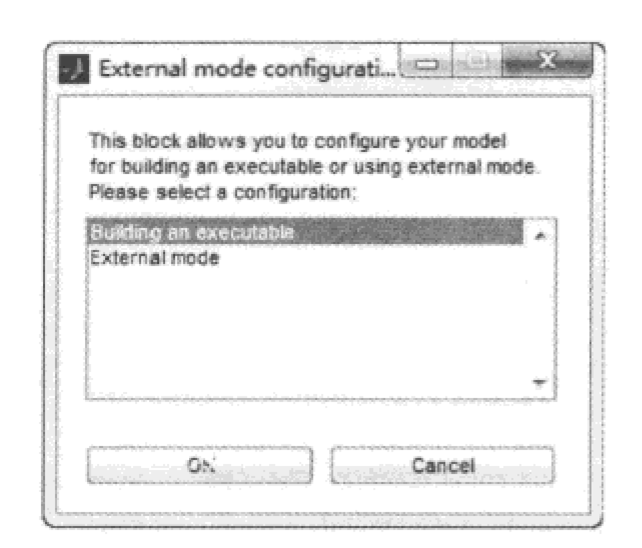


图 6.1.7　Switch External Mode Configuration 设置界面

Switch Target Configuration 模块可将模型设置为一系列预设硬件配置中的某一种，例如 c166_sim、c166cr_hw、c167cs_sim、c167cr_hw 等。

6.2　TASKING EDE for C166

6.2.1　电动机控制模型

本例将重用 5.3.2 节的例子，建模过程不再重复。开发 C166 目标需要 TASKING 专为 C166 设计的 EDE 软件 Classic C166/ST10 Trial Version，下载安装过程可参考 5.3.1 节内容。

在模块库 Simulink\Ports &Subsystems 中找到输入模块与输出模块，替换功能验证模型中的常数与指示灯，并将这些端口的数据类型修改为 uint8，如图 6.2.1 所示。

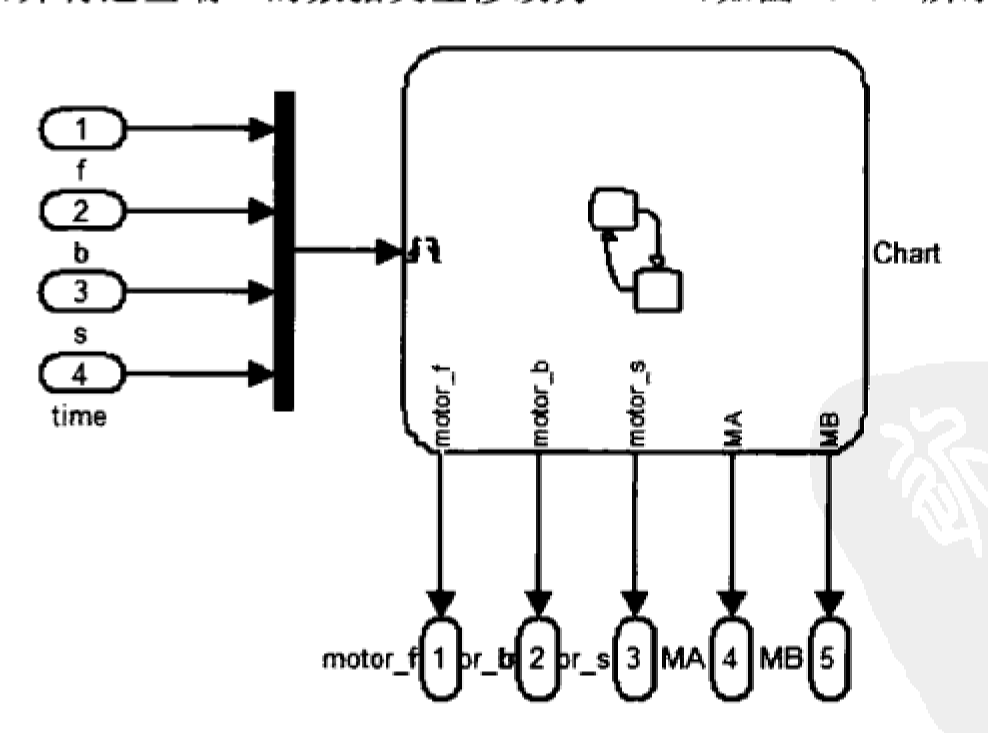


图 6.2.1　代码生成模型

要确保其数据类型和 stateflow 中的数据类和输入/输出端口的数据类型一致，这里仍然推荐使用模型浏览器批量修改，如图 6.2.2、图 6.2.3 所示。

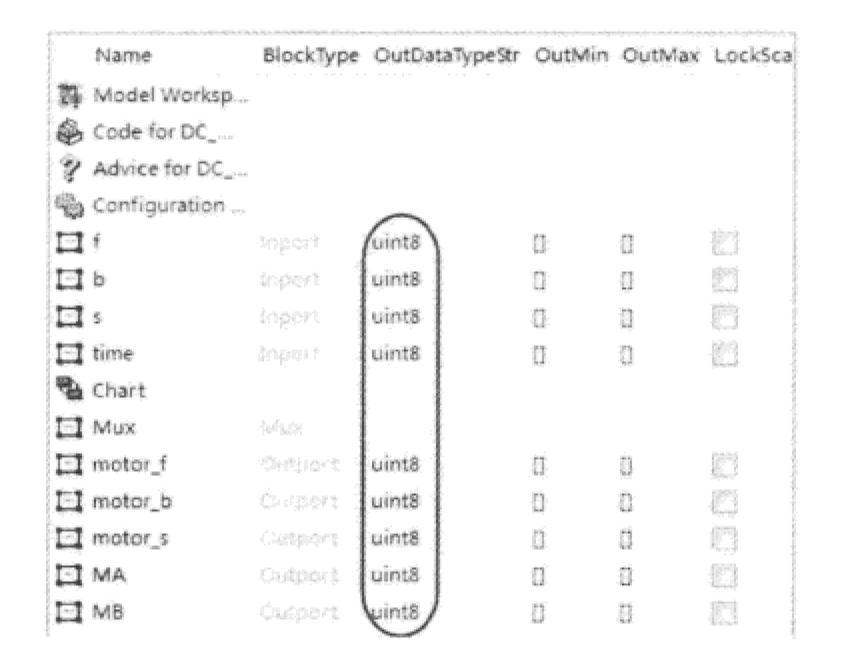


图 6.2.2 修改代码生成模型端口数据类型

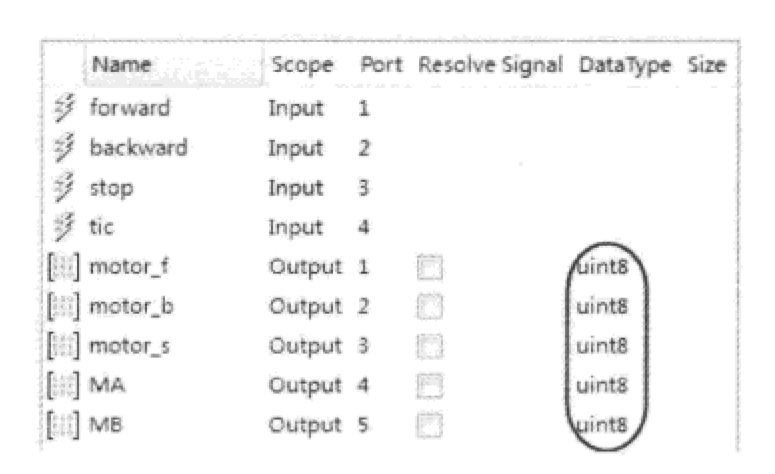


图 6.2.3 修改代码生成模型内参数数据类型

6.2.2 设置 IDE 与模型参数

设置 IDE。选择模型菜单项 Tools→Utilities for Use with TASKING(R) IDE→Add Embedded IDE Link Configuartion to Model…,为模型添加 IDE Link 选项。

选择模型菜单项 Tools→Utilities for Use with TASKING(R) IDE→Target Preferences,选择 C166 选项,设置其目标选项,如图 6.2.4 所示。

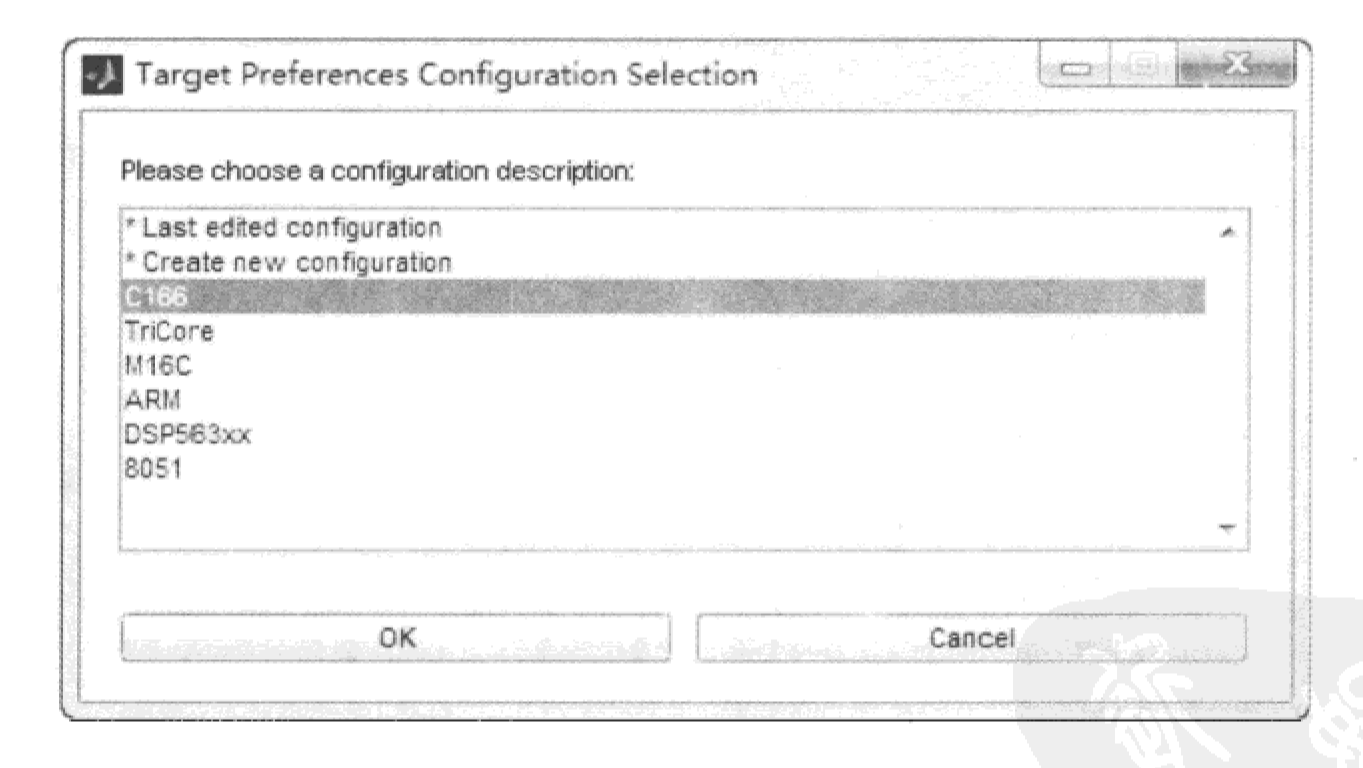


图 6.2.4 选择 C166 选项

选定 C166 目标后,需要在弹出的 Embedded IDE Link Target Preference 对话框中进一步配置所需文件的路径,如图 6.2.5 所示。

指定 CrossViews_Pro_Executable、DOL_File、EDE_Executable 这 3 个文件的位置,与 TASKING 的安装目录有关,例如用 D:\Program Files\TASKING\c166 v8.8r1 替换掉＜ENTER_TASKING_PATH＞,如图 6.2.6 所示。

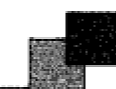




图 6.2.5　设置 IDE Link

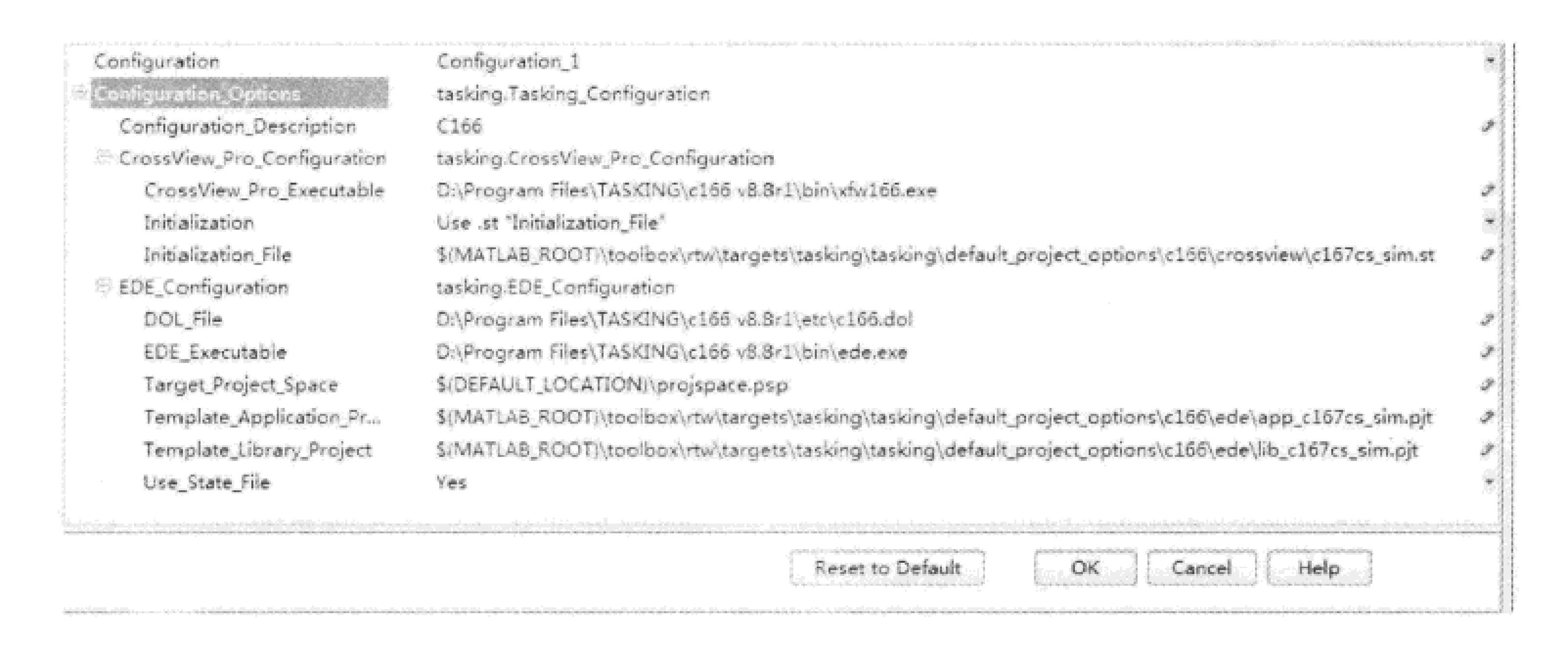


图 6.2.6　修改路径

MATLAB R2010a 版所指定的 TASKING EDE for C166 版本为 v8.7r1，而 TASKING 网站只提供 v8.8r1 试用版软件下载。由于 MATLAB R2010a 不支持 v8.8r1 版软件，因此需对 R2010a 软件中有关 C166 的个别文件做一些修补。请用户登录北京航空航天大学出版社"下载中心"下载经过修改的 EDE 文件，然后替换 matlabroot:\R2010a\toolbox\rtw\targets\tasking\tasking\default_project_options\c166 中的 EDE 文件。

注意：此方法仅供学习本书之用，如果运用于实际项目开发，请读者安装 MathWorks 公司指定的正版 TAKSING EDE 软件。

设置模型参数。打开模型的参数设置对话框，在 Solver Options 界面，设置求解器为定步长离散求解器，在 Solver 下拉列表中选择 discrete 选项、步长使用默认的 auto，也可自行指定，如图 6.2.7 所示。

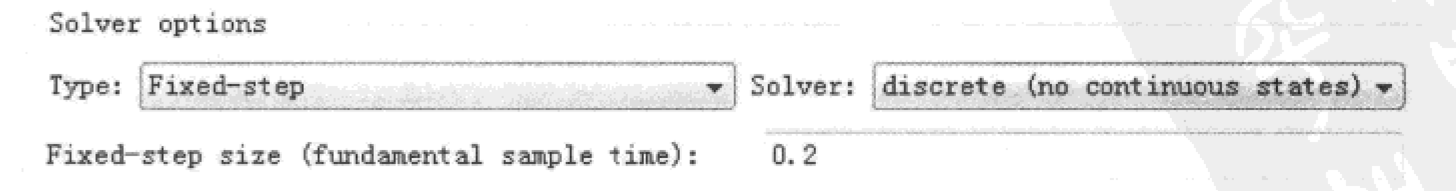


图 6.2.7　求解器设置

在 Hardware Implimentation 界面，设置器件类型为 C16x，如图 6.2.8 所示。

Device vendor: Infineon　Device type: C16x, XC16x
Number of bits　Largest atomic size
char: 8　short: 16　int: 16　integer: Char
long: 32　float: 32　double: 64
native: 16　pointer: 16　floating-point: None

图 6.2.8　选择芯片

在 Real-Time Workshop 界面，设置 TLC 文件为 c166.tlc，如图 6.2.9 所示。

Target selection
System target file: c166.tlc　Browse...
Language: C
Description: Target Support Package (for use with Infineon C166)

图 6.2.9　设置 TLC 文件

在 Report 界面，勾选所有复选框，便于后期检查及跟踪，如图 6.2.10 所示。

Create code generation report　Launch report automatically
Navigation
Code-to-model
Model-to-code　Configure...
Traceability Report Contents
Eliminated / virtual blocks
Traceable Simulink blocks
Traceable Stateflow objects
Traceable Embedded MATLAB functions

图 6.2.10　报告界面设置

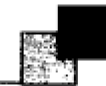
6.2.3　处理器在环测试(PIL)

本实例在 2010b 版中可能会出现一些问题，对于初学者来说不太好解决，因此推荐在 Matlab R2010a 版本中测试，下面的实例演示就是在 2010a 版中完成的。

在 Embedded IDE Link 界面中，目标配置文件应选为 c166，Build action 下拉列表中选择 Create and Build Application Libaray 选项，如图 6.2.11 所示。

Build Configuration
Build action: Create and Build Application Project
Target Preferences Configuration: C166　Edit Configuration
Add build directory suffix
Build directory suffix:

图 6.2.11　IDE 设置



完成上述设置后，在模型的参数设置对话框的 Embedded IDE Link 界面中选择 PIL 验证，如图 6.2.12 所示。

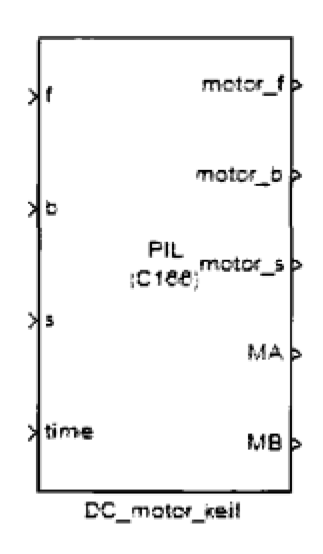
Processor-in-the-Loop (PIL) Verification

☑ Configure model to build PIL algorithm object code

PIL block action: Create PIL block, then build and download PIL application

图 6.2.12 PIL 设置

单击模型工具栏的按钮，生成 PIL 模块，如图 6.2.13 所示。



6.2.13 PIL 模块

用该 PIL 模块替换图 6.2.1 所示的电动机控制模型，建立 PIL 功能验证模型。需要注意的是输入端口的数据类型应设为 uint8，一般和 PIL 模块相匹配。在 Pulse Grnerator 模块后，需添加数据类型转换模块，将 double 转化为 uint8 型，在 Light Bulb 模块前，应添加数据类型转换模块，将 uint8 型数据转换为 double 型，如图 6.2.14 所示。

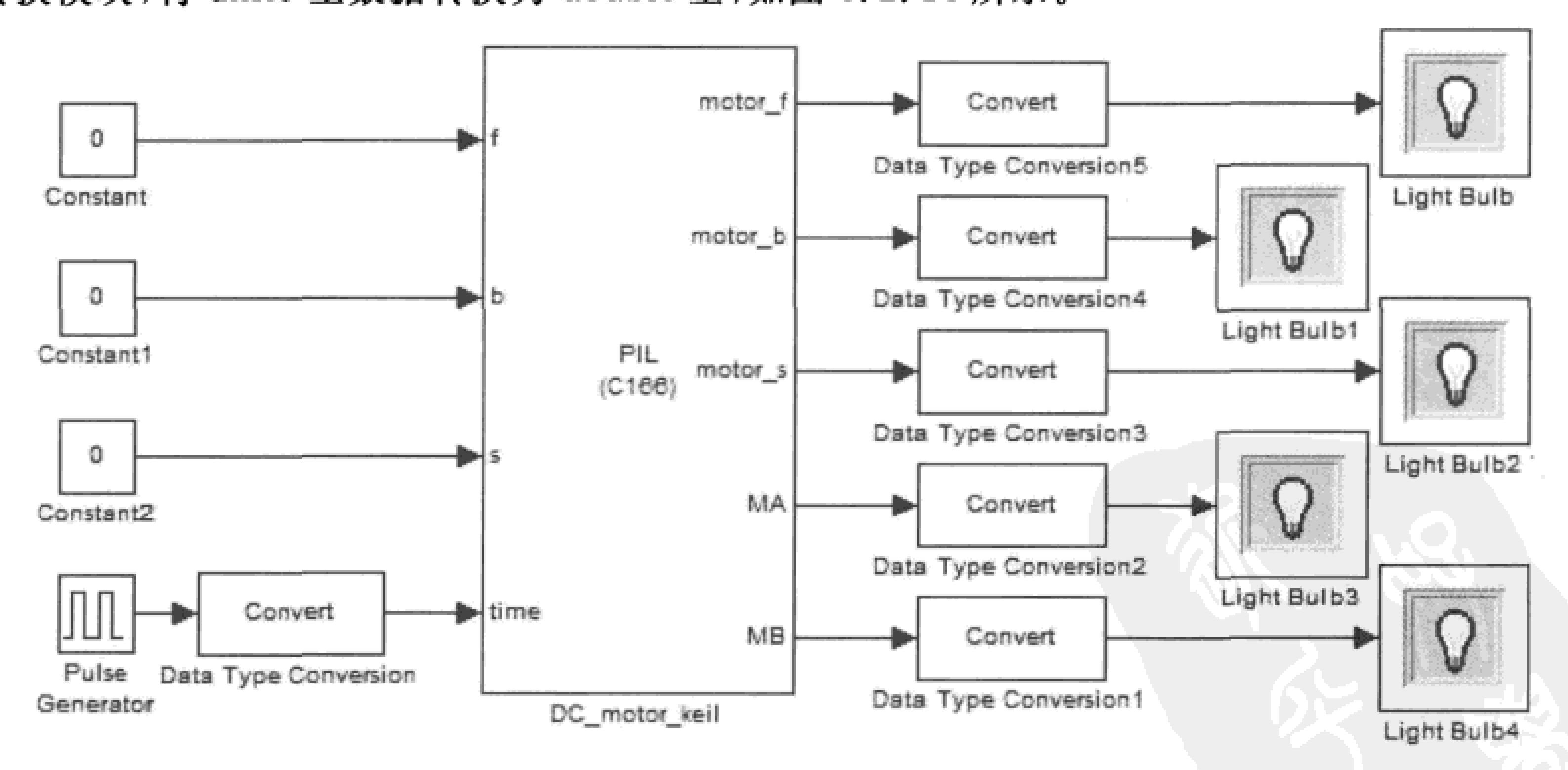


图 6.2.14 处理器在环测试(PIL)

运行 PIL 测试，输出端 Light Bulb 的变化规律与功能验证模型相同。验证了针对具体硬件 C166，自动生成的嵌入式 C 代码的正确性，如图 6.2.15、图 6.2.16 所示。

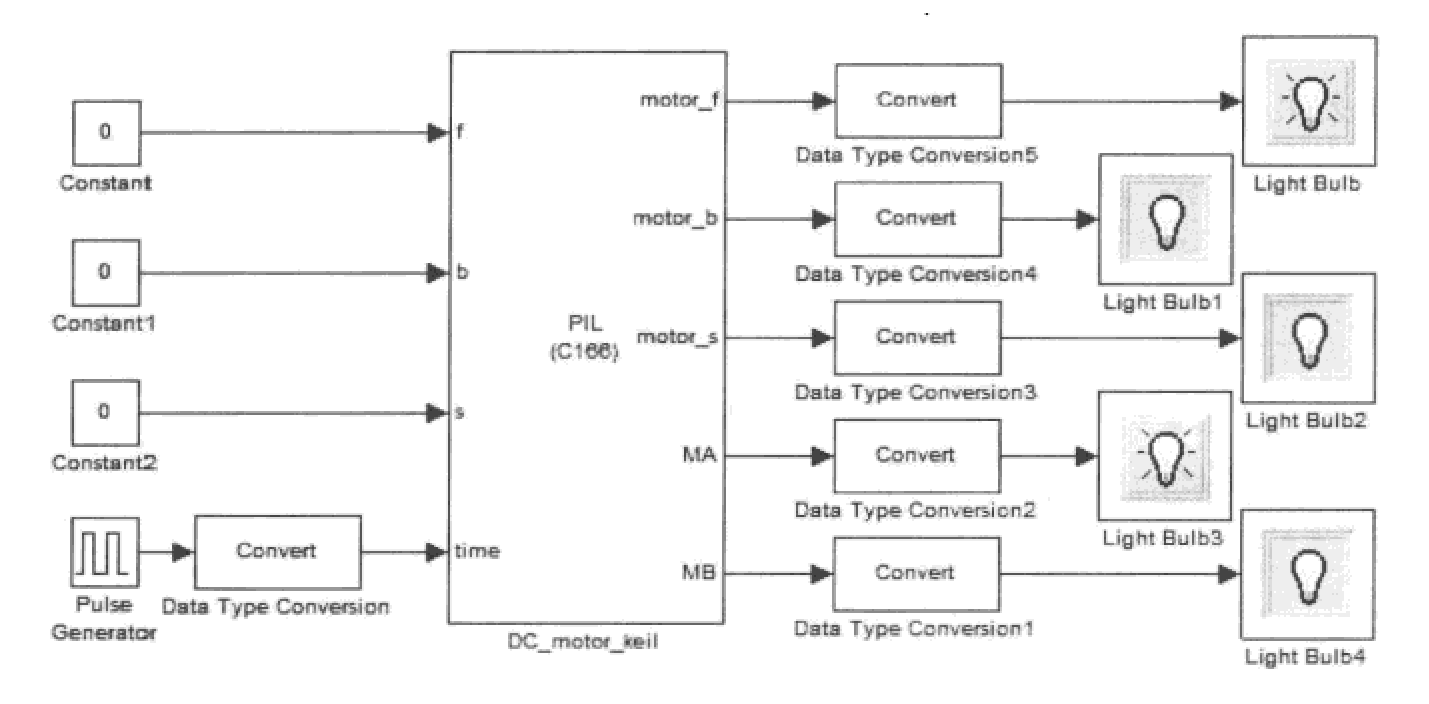


图 6.2.15　仿真结果

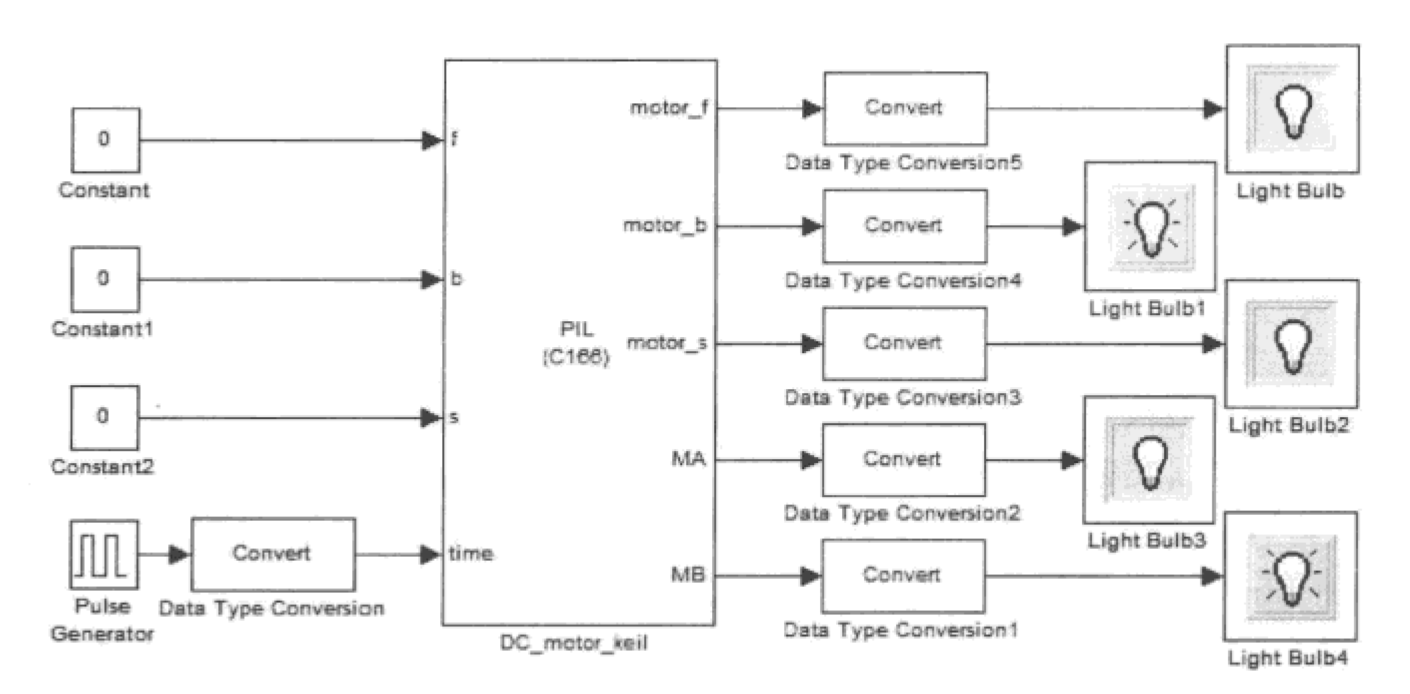


图 6.2.16　仿真结果

同时，在 CrossView Pro C166/ST10 中可以观察到模拟处理器的堆栈，存储器等值，如图 6.2.17 所示。

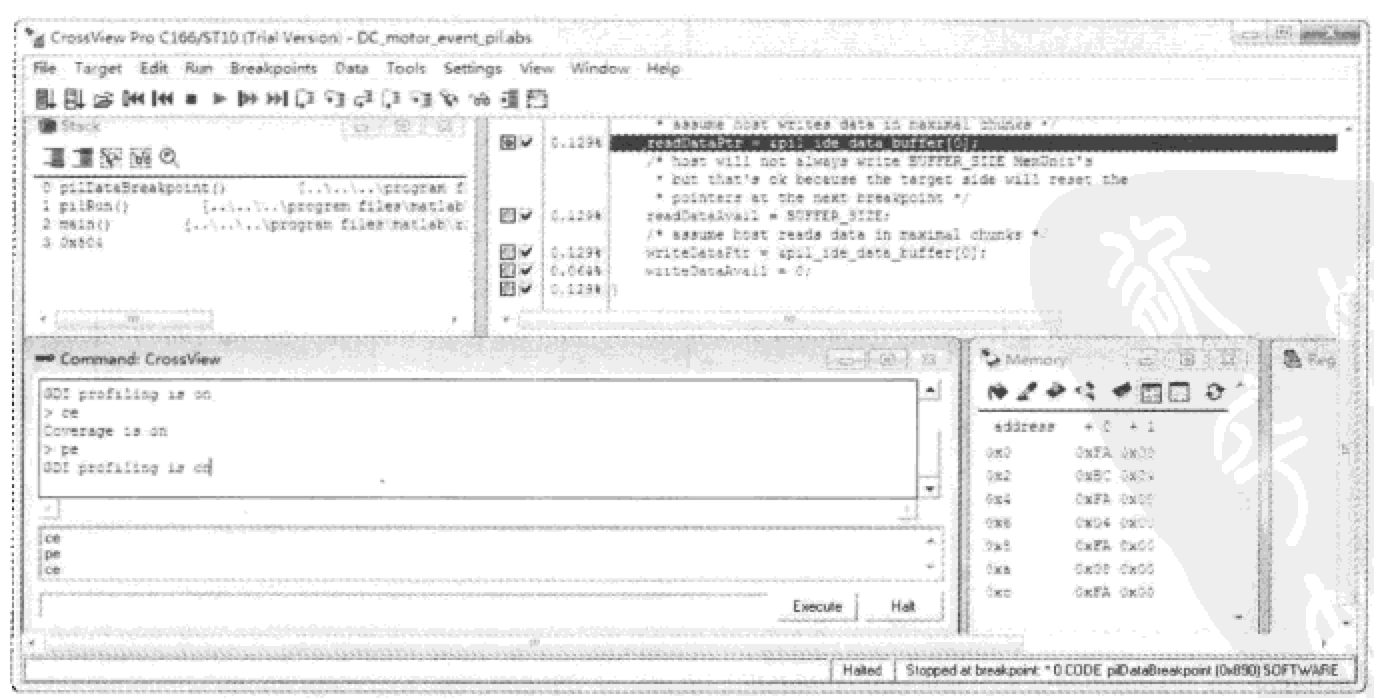
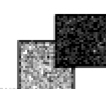


图 6.2.17　CrossView Pro C166/ST10 界面



6.2.4 代码的自动生成

再次打开模型参数设置对话框，在 Embedded IDE Link 界面中将 PIL 验证关闭，单击模型工具栏的按钮，生成模型代码并给出了代码生成报告，如图 6.2.18 所示。

Back | Forward

Contents

Summary
Subsystem Report
Code Interface Report
Traceability Report

Generated Files

[-] **Main file**

c166_main.c

[-] **Model files**

DC_motor_event.c
DC_motor_event.h
DC_motor_event_private.h
DC_motor_event_types.h

[+] **Shared Utility files (2)**

```
17
18  #include <stddef.h>                    /* Defines NULL */
19  #include "reg167cs.h"                  /* For selected C166 architecture */
20  #include "rtwtypes.h"
21  #include "DC_motor_event.h"            /* Model's header file */
22
23  /* Control the behaviour of the model when overruns occur */
24  #define BASE_RATE_MAX_OVERRUNS         5
25
26  /* Trap numbers used for each task */
27  #define TASK_0_TRAP                    0x79
28
29  /* Assembler function that initializes the task dispatcher. */
30  extern void InitDispatcher(void);
31
32  /* Allow assembler function that calls model tasks to be called
33   * directly instead of as the timer interrupt service routine. */
34  extern void dispatch_p(void);
35  extern volatile uint16_T OverrunCounter;
36
37  /* rt_OneStep is called by the dispatcher. If no base rate overruns occur
38   * rt_OneStep will be called once per sample period. In the event that overruns
39   * do occur, the dispatcher ensures that only one instance of rt_OneStep is
40   * is running at any one time, in this case rt_OneStep must "catch up" by
41   * updating the base rate as many times as necessary within the same
42   * function call until all base rate overruns are cleared.
43   */
44  _interrupt ( TASK_0_TRAP ) void rt_OneStep( void )
45  {
46    /* Check if an error status has been set by an overrun or generated code */
47    if (rtmGetErrorStatus(DC_motor_event_M) != NULL) {
48      return;
49    }
50
51    while (OverrunCounter > 0) {
52      /* Re-enable timer interrupt. To achieve this the CPU priority
53       * level is lowered so that timer interrupts are acknowledged
54       * by the CPU
55       */
```

图 6.2.18　代码生成报告

系统会自动打开 TASKING EDE 开发环境，创建工程，并编译调试，最后生成 HEX 文件，如图 6.2.19 所示。

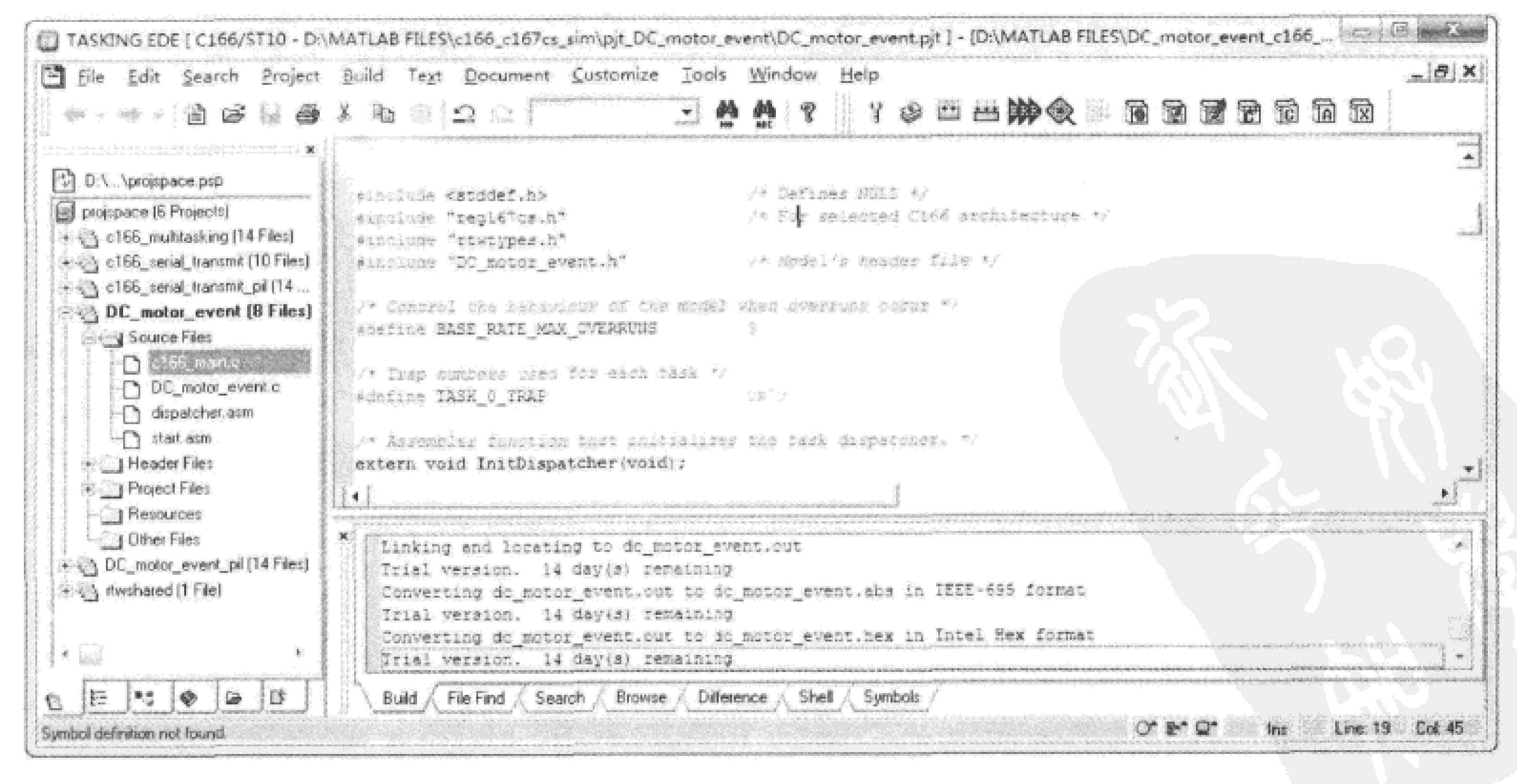


图 6.2.19　TASKING 工程

生成代码后，利用 MiniMon 软件将目标板和 PC 通过虚拟串口连接，可以直接向目标板烧写程序。

MiniMon 软件可在 http://www.infineon.com/ 中下载。打开网页，单击 Product Categories 中的 Microcontrollers→Development Tools，Software and Kits，如图 6.2.20 所示。



图 6.2.20　Microchip 网页

在新页面中选择 C166/XC166 Development Tools and Software 即可下载 Minimon。

按提示安装好 MiniMon，接下来在 MATLAB 命令行中输入 c166utils，在弹出的 Target Support Package Utilities for Use with C166 对话框中进行配置，如图 6.2.21 所示。

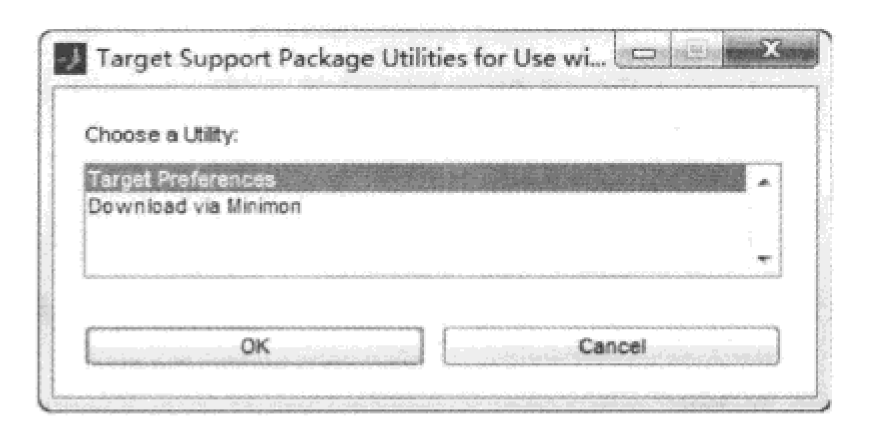


图 6.2.21　目标支持工具选择

选择 Target Perferences 选项，然后在 Target Support Package Target Preferences 对话框中指定 MiniMon 的路径，如图 6.2.22 所示。

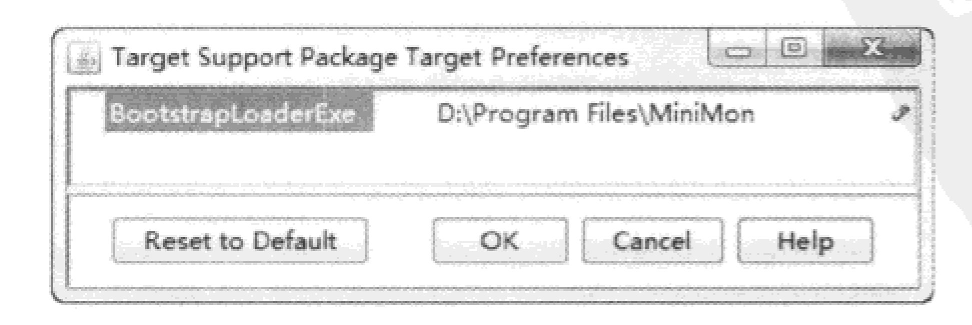
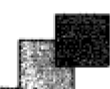


图 6.2.22　指定路径



如果用户使用的不是指定的目标开发板，则还需要根据硬件的实际情况对代码进行一定的调整。

在 TASKING 工程的 Source Files 中打开 c166_main.c，如图 6.2.23 所示。可以发现代码还需要用户根据实际的硬件连接情况对代码进行一定的调整，例如设置输入/输出端口，对主函数进行微调等，具体可参考 5.3.2 节的内容。

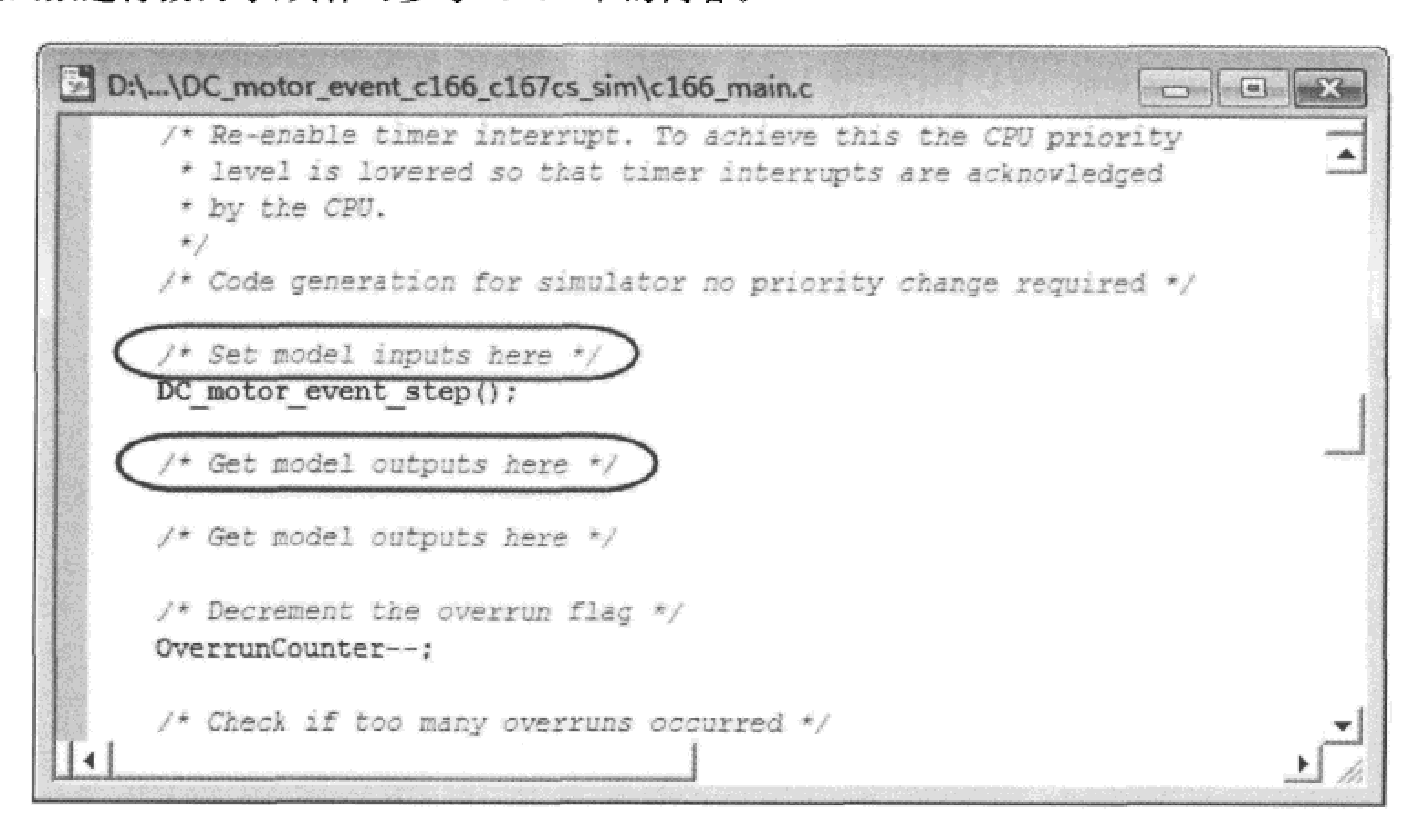


图 6.2.23 修改代码

代码修改完毕后单击 TASKING EDE 工具栏按钮，打开工程选项窗口，如图 6.2.24 所示，在 Linker→Output Format 界面中，勾选 Intel Hex records 复选框。

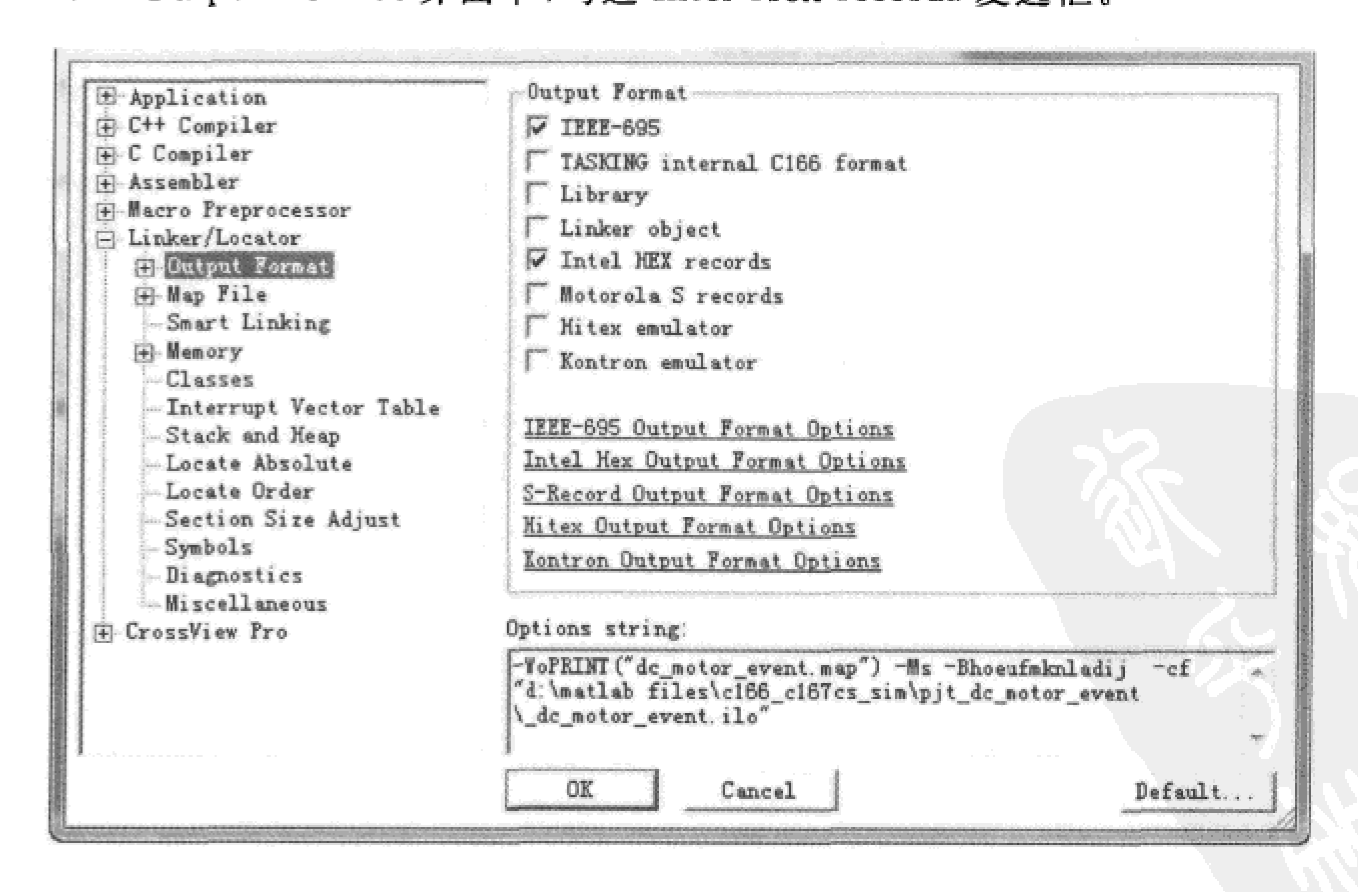


图 6.2.24 设置输出 HEX 文件

再单击工具栏按钮，重编译工程，如图 6.2.25 所示，窗口下部的信息显示已成功生成 HEX 文件，可以加载到硬件上运行了。

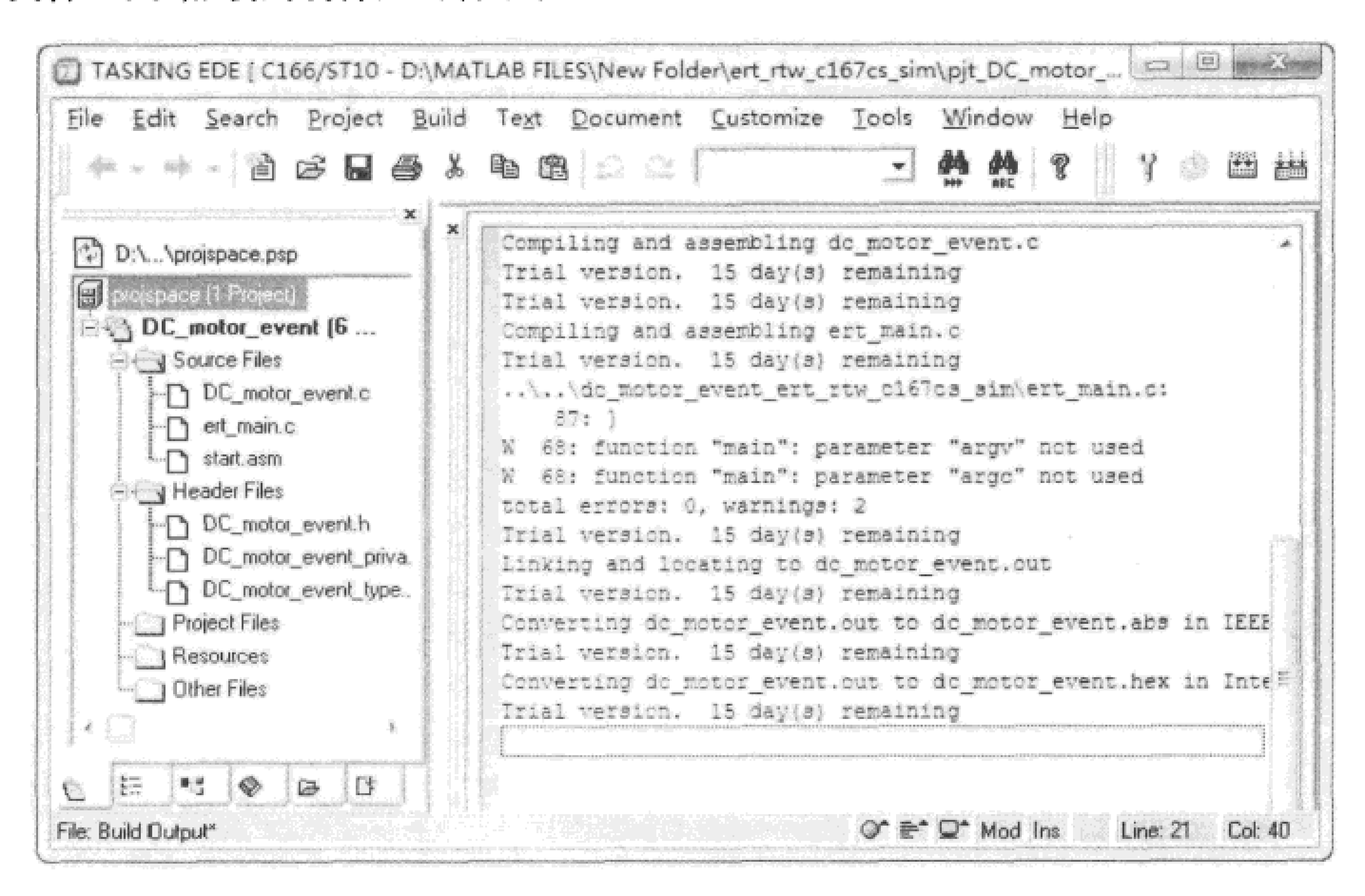


图 6.2.25 编译 TASKING 工程



第7章

基于 Simulink 模块的 dsPIC 单片机开发

前面几章讲述了模型代码加手工代码的方式，开发单片机的新技术，不过这种方法也存在重复编写器件配置代码的问题。为了加快项目的开发速度和避免不必要的重复劳动，Microchip 公司针对 dsPIC30 和 dsPIC33 DSC 等器件，提供了一套接口兼容的配置和运行时外设模块集——MATLAB Plug-in blockset。它可以使 MATLAB/SIMULINK/Stateflow 与 MATLAB Plug-in blockset/MPLAB IDE 无缝连接，利用 Real-time Workshop Embedded Coder 代码生成工具，自动生成应用的实时嵌入式 C 代码。可极大地提高工作效率和降低开发成本，同时也可降低了运用 dsPIC3x DSC 器件开发工程师的门槛。下面对 MATLAB Plug-in blockset 作一简单介绍。

模块集的主要特性：

(1) 模块适用于 dsPIC3x 系列的所有外设。

(2) 可通过"cCall"模块方便地集成经过验证 C 代码。

(3) 从 MATLAB 环境中构建环境配置。

版本 2.0 的新增功能：

(1) 支持 dsPIC30F 系列器件。

(2) 电动机控制算法库。

(3) 缺陷修正和改进。

(4) 演示版——免费评估版。

(5) 与 MPLAB 的 MATLAB 插件更好地集成在一起。

(6) 与从 R2007a 到 R2009b 的所有 MATLAB 版本兼容，不过也支持 R2010a、R2010b 版(作者安装的版本是 R2010b)。模块适用于 dsPIC3x 系列的所有外设。

本章的主要内容：

- MPLAB 开发工具简介。
- dsPIC 外围驱动模块介绍及应用。
- 无对应驱动模块时的应用。

7.1 MPLAB 嵌入式开发环境及工具

MPLAB IDE 是 Microchip 公司开发的基于 Windows 操作系统的集成开发环境，适用于 PICmicro MCU 系列和 dsPIC 数字信号控制器的开发。同时，MPLAB IDE 还将其他 Microchip 工具集和第三方软件集成到一个图形用户界面之中。其主要功能如下：

(1) 使用内置的编辑器创建和编辑源代码。

(2) 汇编、编译和链接源代码。

(3) 通过使用内置的软件模拟器观察程序流程，或者使用在线仿真器或在线调试器以实时方式观察程序流程来调试可执行逻辑。

(4) 用软件模拟器或仿真器进行时序测量。

(5) 查看 Watch 窗口中的变量。

MATLAB/Simulink device blocksets 向 Simulink 模型库中添加了 dsPIC 模块。这些模块既可以单独使用，也能和 Simulink 模块混合建模，并通过 Real-time Workshop Embedded Coder 自动生成嵌入式实时 C 代码，再经 MPLAB IDE 实现嵌入式应用的基于模型设计的开发。

MPLAB C30 Compiler 是针对 16 位 dsPIC 芯片 dsPIC30 和 dsPIC33 设计的高度优化的编译器，通过 MPLAB C30 Compiler 或其他第三方编译器可以用 C 语言实现 dsPIC 开发。

7.1.1 软件的下载和安装

用户在浏览器地址栏中输入网址：http://www.microchip.com/stellent/idcplg? IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002，打开 MPLAB 下载网页，如图 7.1.1 所示。

MPLAB Software

Downloads	Associated Files and Release Notes
MPLAB IDE v8.56 Full Release Zipped Installation	Release Notes for MPLAB IDE v8.56
MATLAB Device Blocks for dsPIC DSCs	

Downloads

Title	Date Published	Size	D/L
MPASM/MPLINK User's Guide	4/8/2009 3:52:41 PM	2896 KB	
MPLAB Assembler, Linker and Utilities for PIC24 MCUs and dsPIC DSCs User's Guide	1/26/2010 10:16:32 AM	1981 KB	
MPLAB IDE User's Guide	1/20/2009 12:09:31 PM	4232 KB	
The MPLAB IDE Debug Tool API	5/13/2010 5:16:00 PM	171 KB	

图 7.1.1　MPLAB 下载网页

在网页下方单击链接 MPLAB IDE v8.56 Full Release Zipped Installation 和 MATLAB Device Blocks for dsPIC DSCs 即可下载 MPLAB IDE 和 MATLAB/Simulinkdevice blocksets。

用户如果使用过 MPLAB 的较早版本，或将 V8.53 版本安装在非默认路径下，请首先将其卸载，不然在后续使用过程中，系统有可能无法定位到需要的文件。

单击 windows 开始菜单，选择控制面板→卸载程序，在列表中选择 MPLAB Tools V8.53 选项，单击“卸载”按钮，将其完全移除，如图 7.1.2 所示。

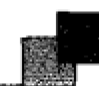




图 7.1.2　卸载早期版本的 MPLAB

1. 安装 MPLAB IDE

打开 MPLAB IDE 安程序，按照安装向导的提示进行安装，在 Setup Type 对话框中尽量点选 Complete 单选按钮，如图 7.1.3 所示。

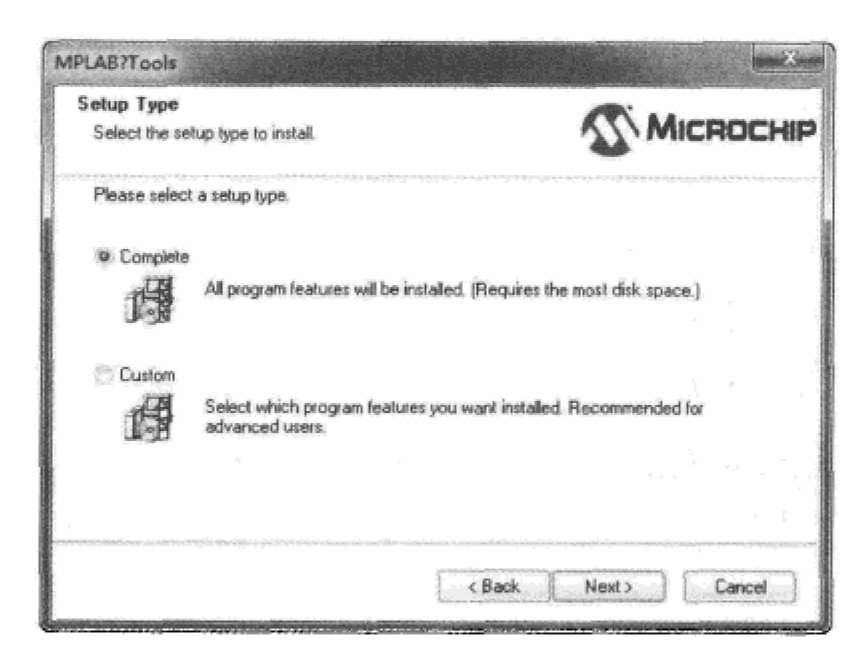


图 7.1.3　MPLAB 安装界面

如果点选了 Custom 单选按钮，则在 Select Features 对话框中应选择 MPLAB IDE Tools 的 MATLAB 选项，以便能够使用 MATLAB Plug-in，如图 7.1.4 所示。

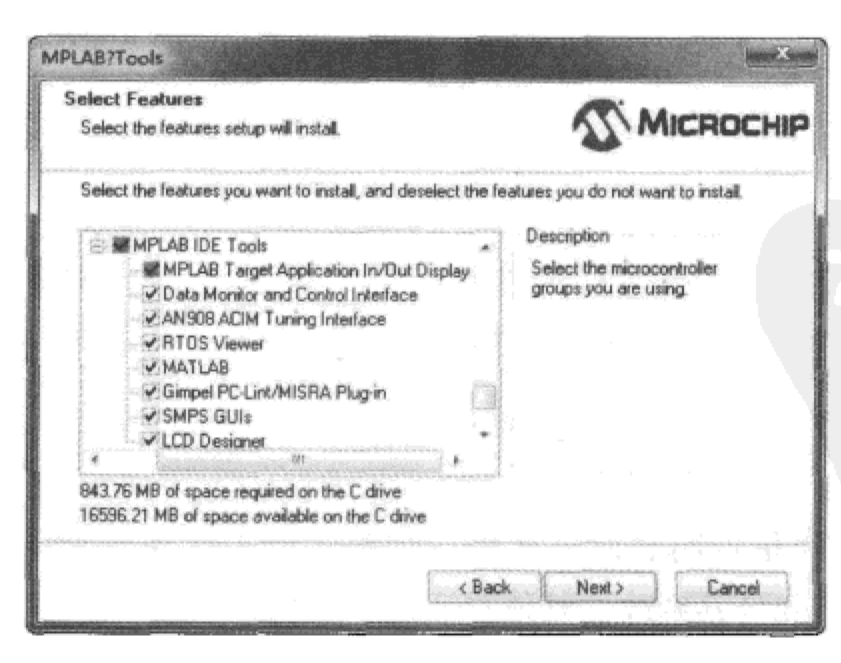


图 7.1.4　MPLAB 安装界面

在 Choose Destination Location 对话框中选用默认路径，不要作任何修改。其他对话框不用作修改，单击“下一步”按钮，如图 7.1.5 所示。

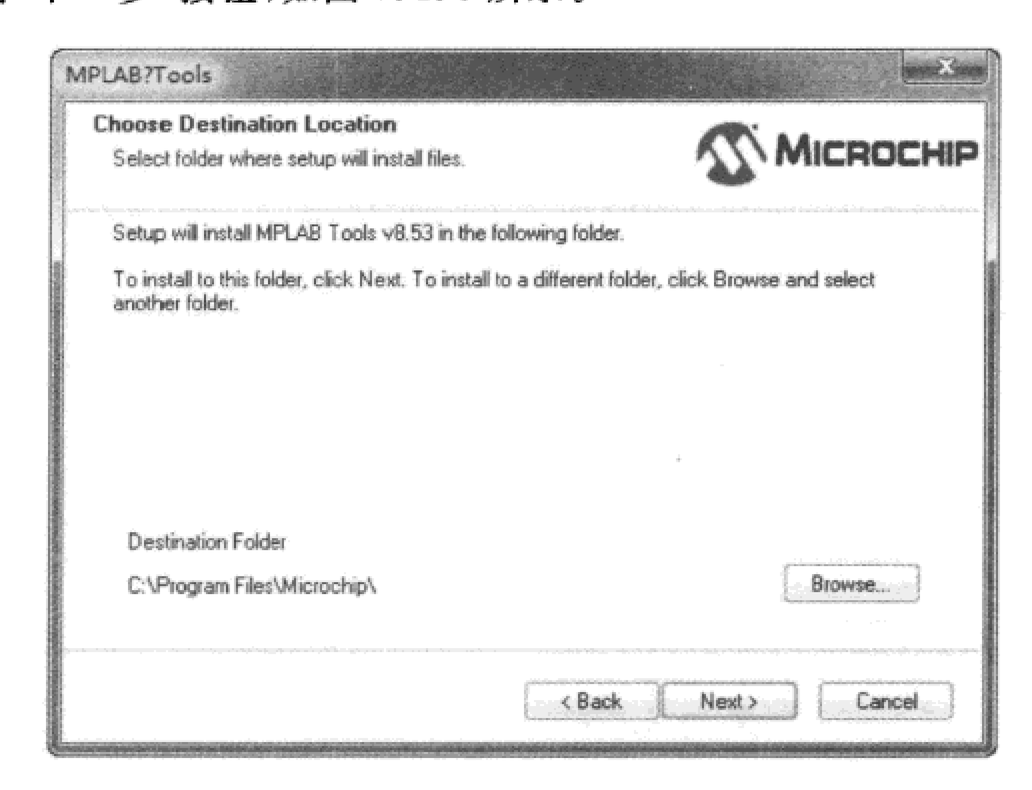


图 7.1.5　MPLAB 安装界面

2. 安装 C30 Compiler

打开 C30 Compiler 安装程序，按照安装向导的提示进行安装，在 Setup Type 对话框中选择 Complete 选项，如图 7.1.6 所示。

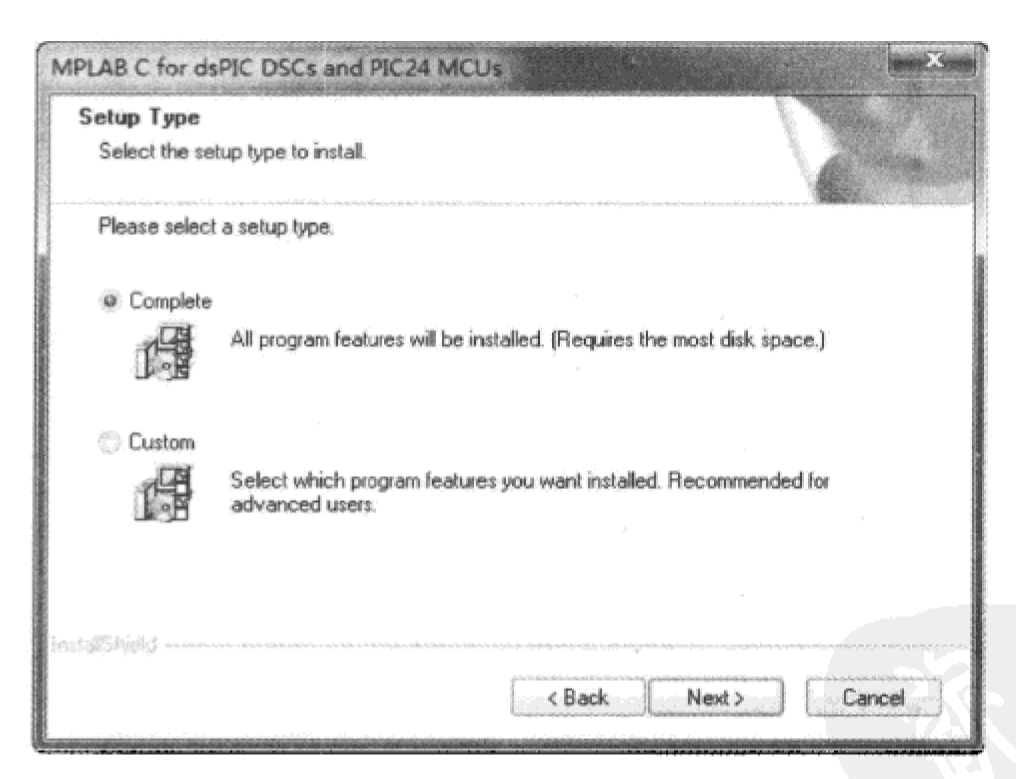
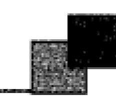


图 7.1.6　C30 编译器安装界面

在 Choose Destination Location 对话框中选用默认路径，不要作任何修改，如图 7.1.7 所示。

下面安装程序会询问是否同意修改环境变量和注册表以便 MPLAB IDE 能够识别到 C30 Compiler，单击“是”按钮，允许其修改。其他对话框不用作修改，单击“下一步”按钮，如图 7.1.8、图 7.1.9 所示。



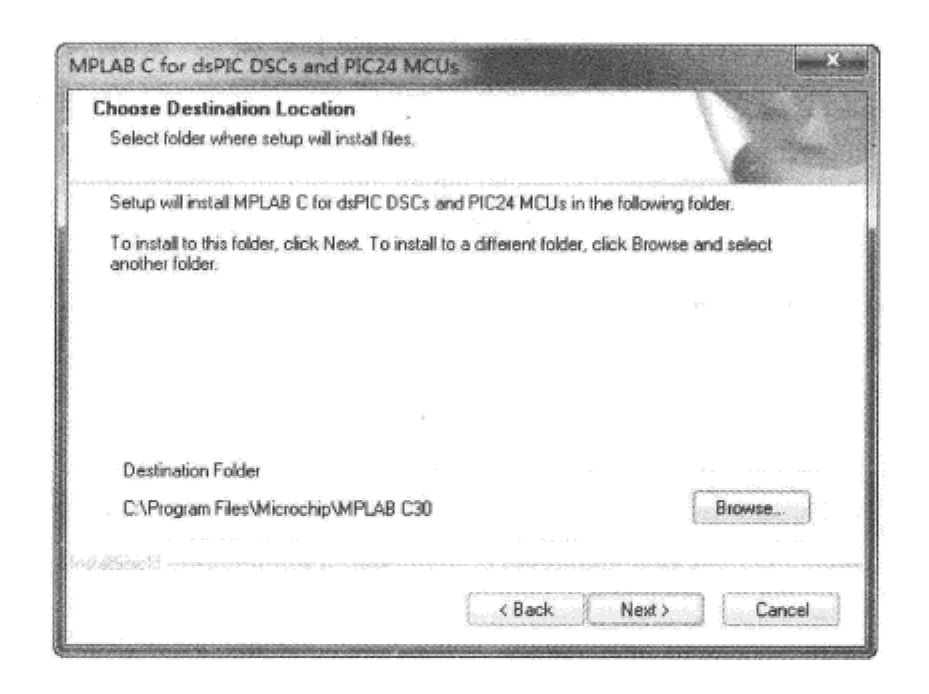


图 7.1.7　C30 编译器安装界面

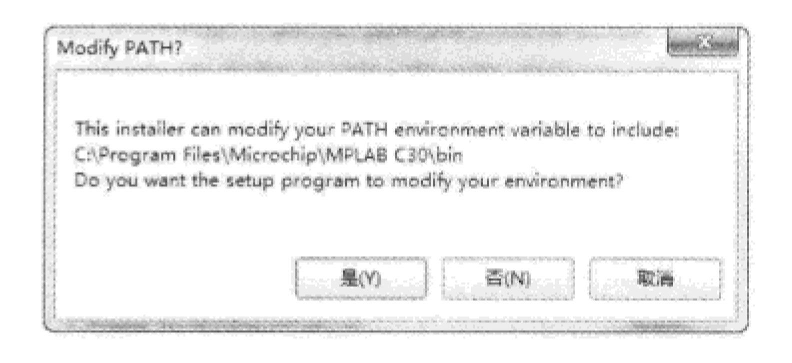


图 7.1.8　C30 编译器安装界面

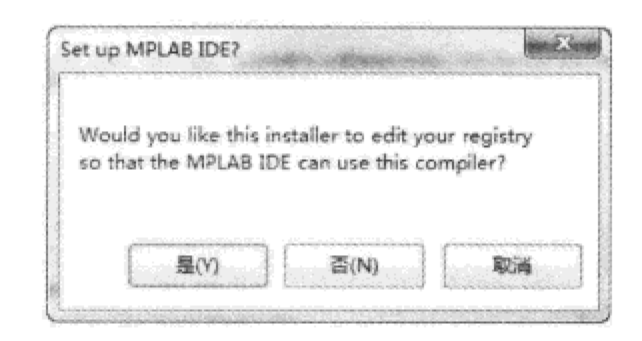


图 7.1.9　C30 编译器安装界面

3. 安装 MATLAB-Blocksets

此处下载的 MATLAB Blockset 是 V2.10 版，如果用户机器中安装了其他版本的 MATLAB Blockset，有可能和 V2.10 版间会产生一些冲突，这里仅就 V2.10 版的 MATLAB Blockset 进行讨论。

打开 MATLAB Blocksets 安程序，按照安装向导的提示进行安装，在 Choose Destination Location 对话框中选用默认路径，不要作任何修改，如图 7.1.10 所示。

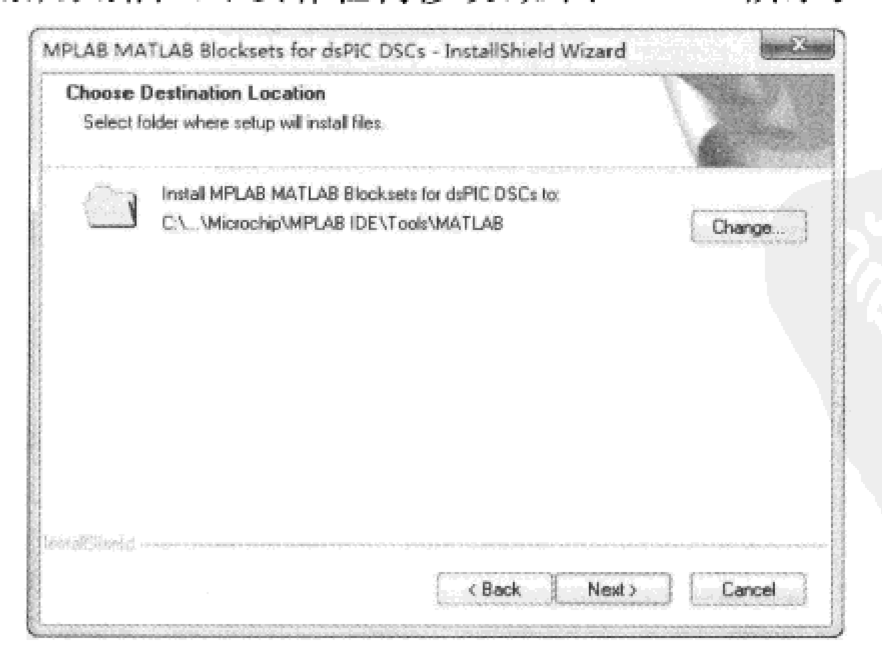


图 7.1.10　MATLAB-Blocksets 安装界面

由于这是较早期的 MATLAB Blocksets,其支持的最高版本 MATLAB 是 R2009b,但对 R2010b 版也仍然兼容,此处勾选 R2009b 复选框。其他对话框不用作修改,单击"下一步"按钮,如图 7.1.11 所示。

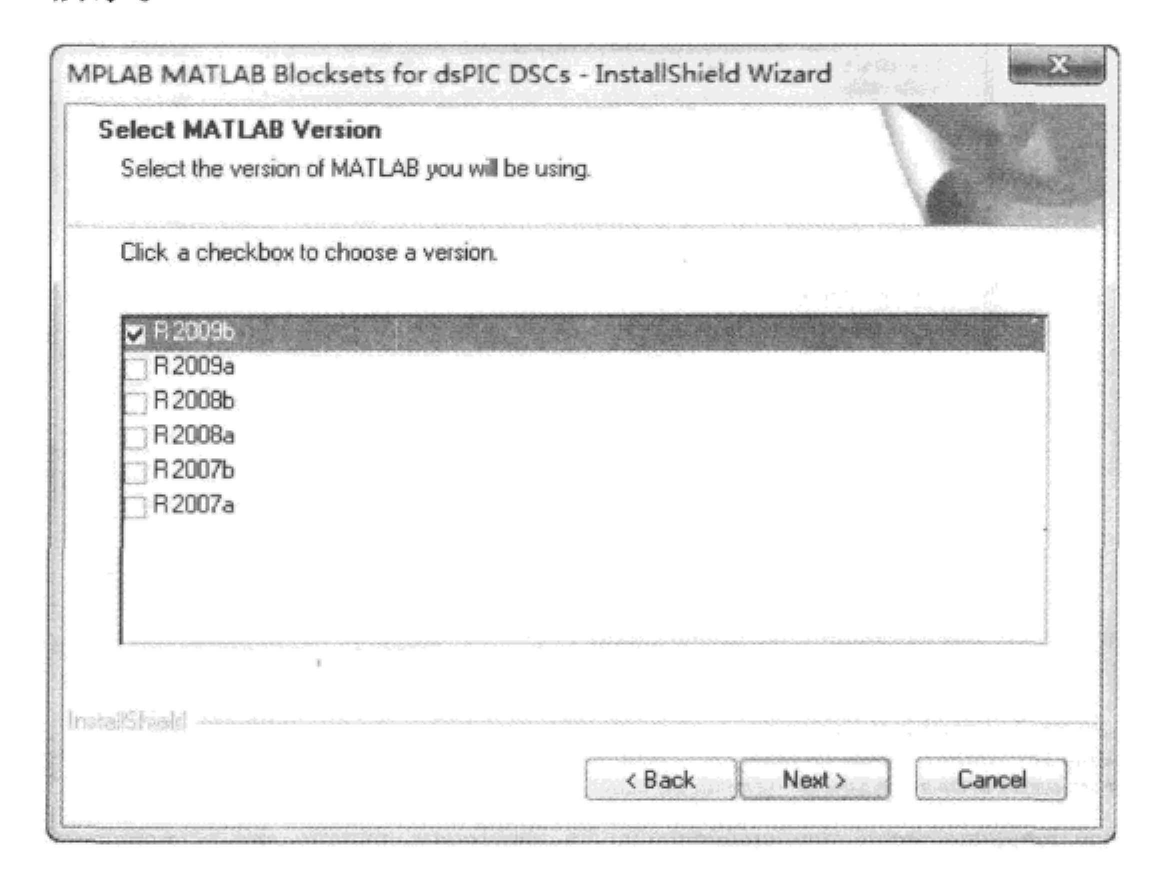


图 7.1.11 MATLAB-Blocksets 安装界面

安装完 MATLAB Blockset 后,dsPIC 模块并没有立即添加到 Simulink 模块库中,用户还需要在 MATLAB 环境中进行添加。

打开 MATLAB,并将当前目录(current folder)设置为 Microchip root:\MPLAB IDE\Tools\MATLAB\dsPIC_Matlab,在当前目录下的文件列表中可以发现一个 Install_dsPIC_Matlab.p 文件,如图 7.1.12 所示。

在左侧的文件列表中右击该文件,并在弹出的右键菜单中选择 Run 选项,如图 7.1.13 所示。

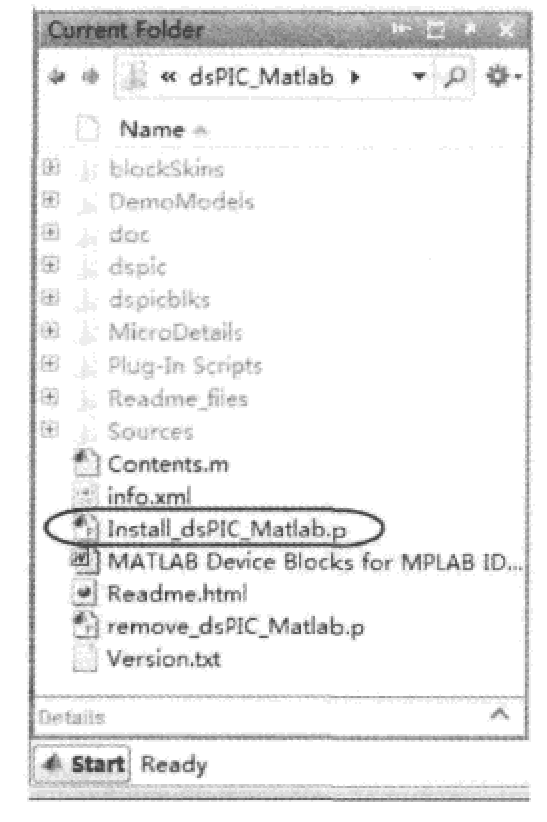


图 7.1.12 定位到安装目录

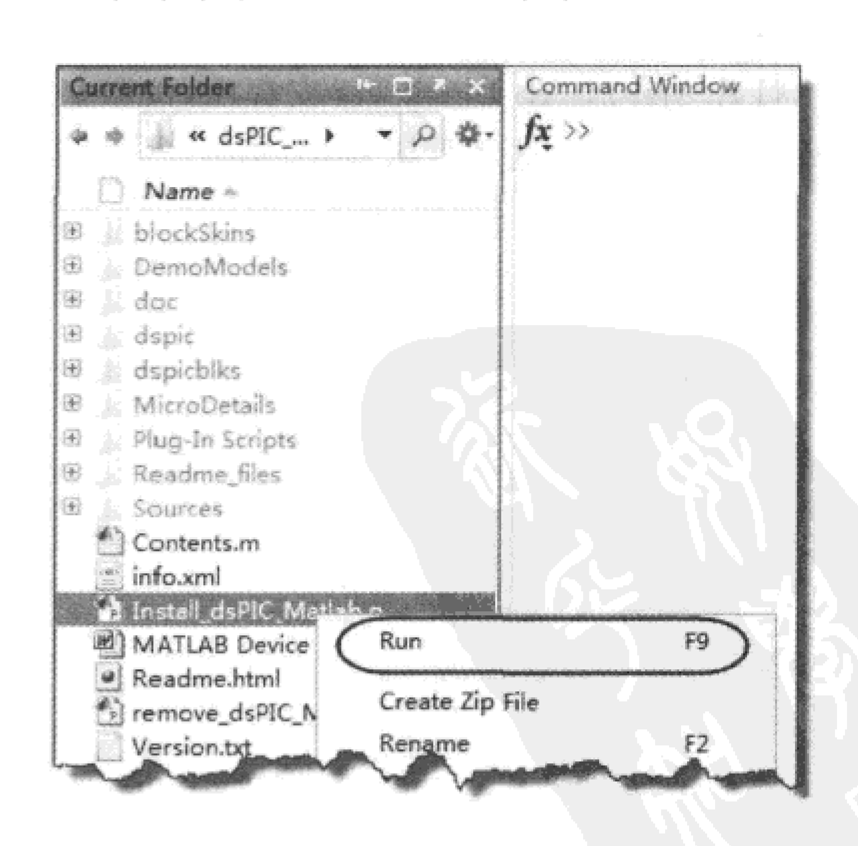
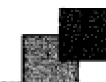


图 7.1.13 运行 Install_dsPIC_Matlab.p 文件



命令行中会出现如下信息：

```
-------------------------
INSTALLING Target for Microchip(TM) dsPIC 2.10
-------------------------
Target for Microchip(TM) dsPIC 2.10 is Installed and ready to use.
```

这时再打开 Simulink 模块库浏览器，就可以看到安装的模块了，如图 7.1.14 所示。

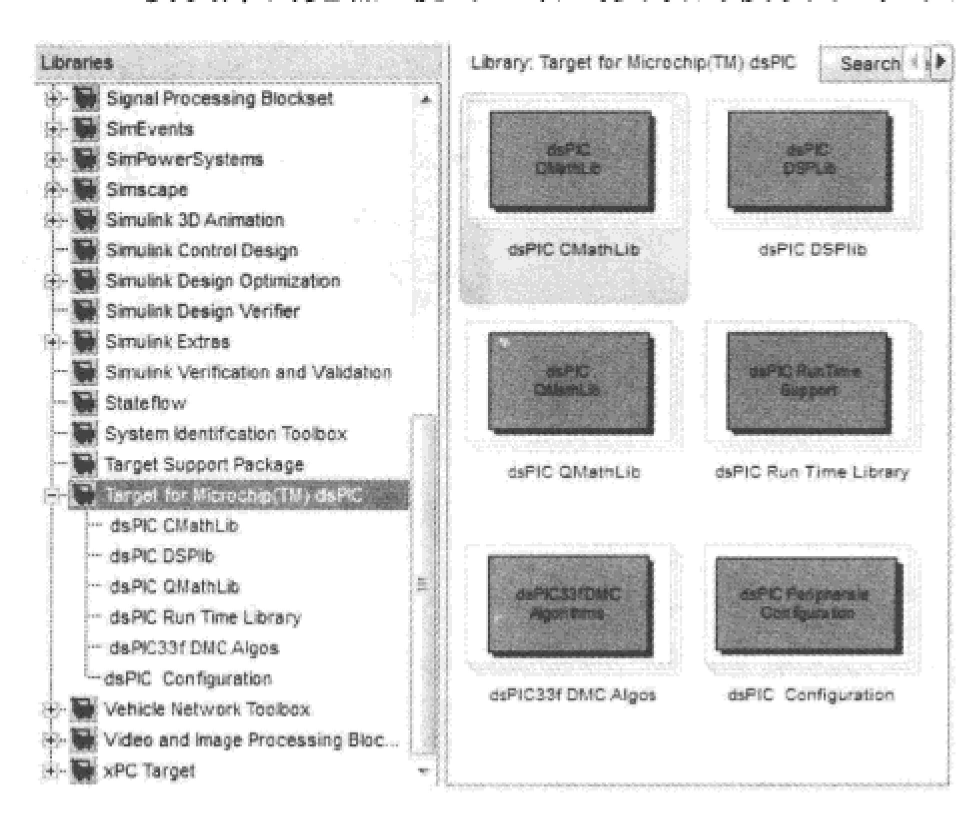


图 7.1.14 dsPIC 模块库

7.1.2 利用 MPLAB IDE 及 Proteus VSM 进行虚拟硬件调试

MPLAB IDE 集成了大量的调试工具，既包括 MPLAB ICD 等仿真器，也能使用 Proteus 软件进行虚拟硬件调试。

打开 MPLAB IDE，选择菜单栏的 Debugger→Select Tool→Proteus VSM 即可在 MPLAB IDE 中打开 Proteus VSM 界面，如图 7.1.15、图 7.1.16 所示。

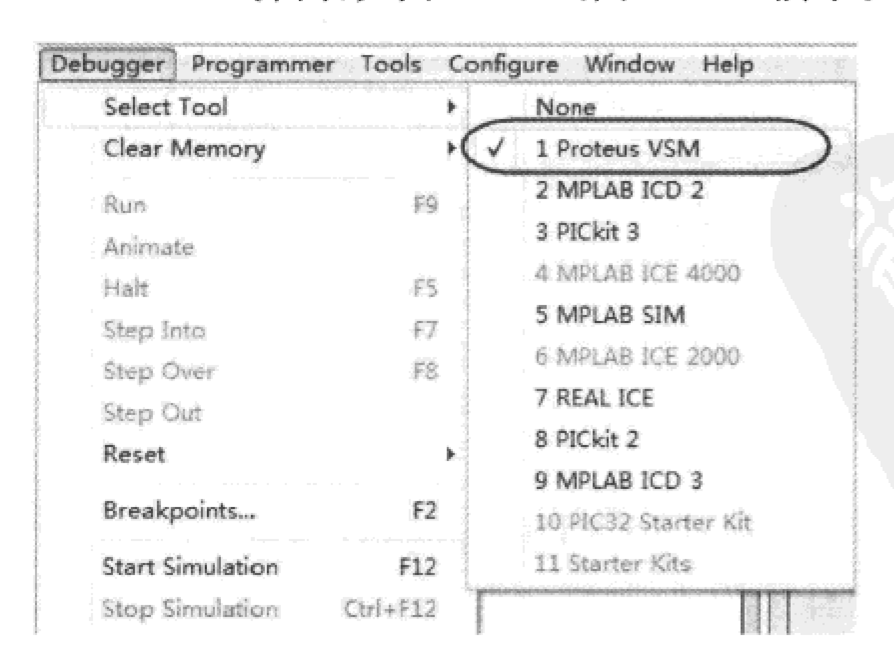


图 7.1.15 选择 Proteus VSM 调试工具

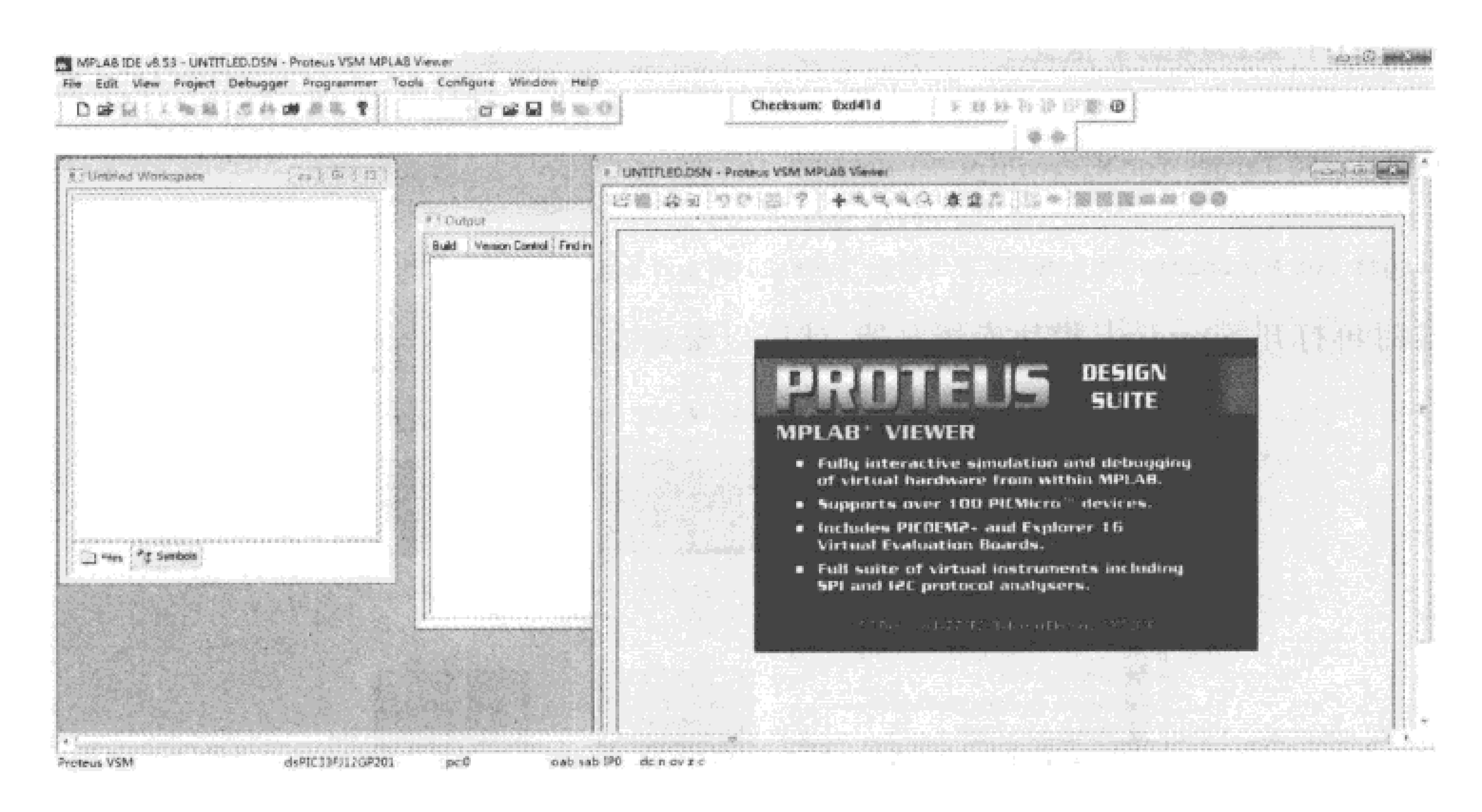


图 7.1.16 调试界面

1. 绘制原理图

作者目前使用的 Proteus 7.7 版本，仅支持 dsPIC33 系列中的一部分芯片，不过 Labcenter 公司即将推出的 Proteus 8.0 版本将会支持更多 dsPIC33 芯片，开拓虚拟硬件测试范围。

本例利用 dsPIC33FJ12GP201 芯片搭建一个能够点亮发光二极管的最简系统，演示如何在 MPLAB IDE 中使用 Proteus VSM 进行虚拟硬件调试。

参考第 5.1 节的内容，不难搭建出图 7.1.17 所示的原理图。图中仅包含驱动电路工作所必须的电源、晶振和复位电路。

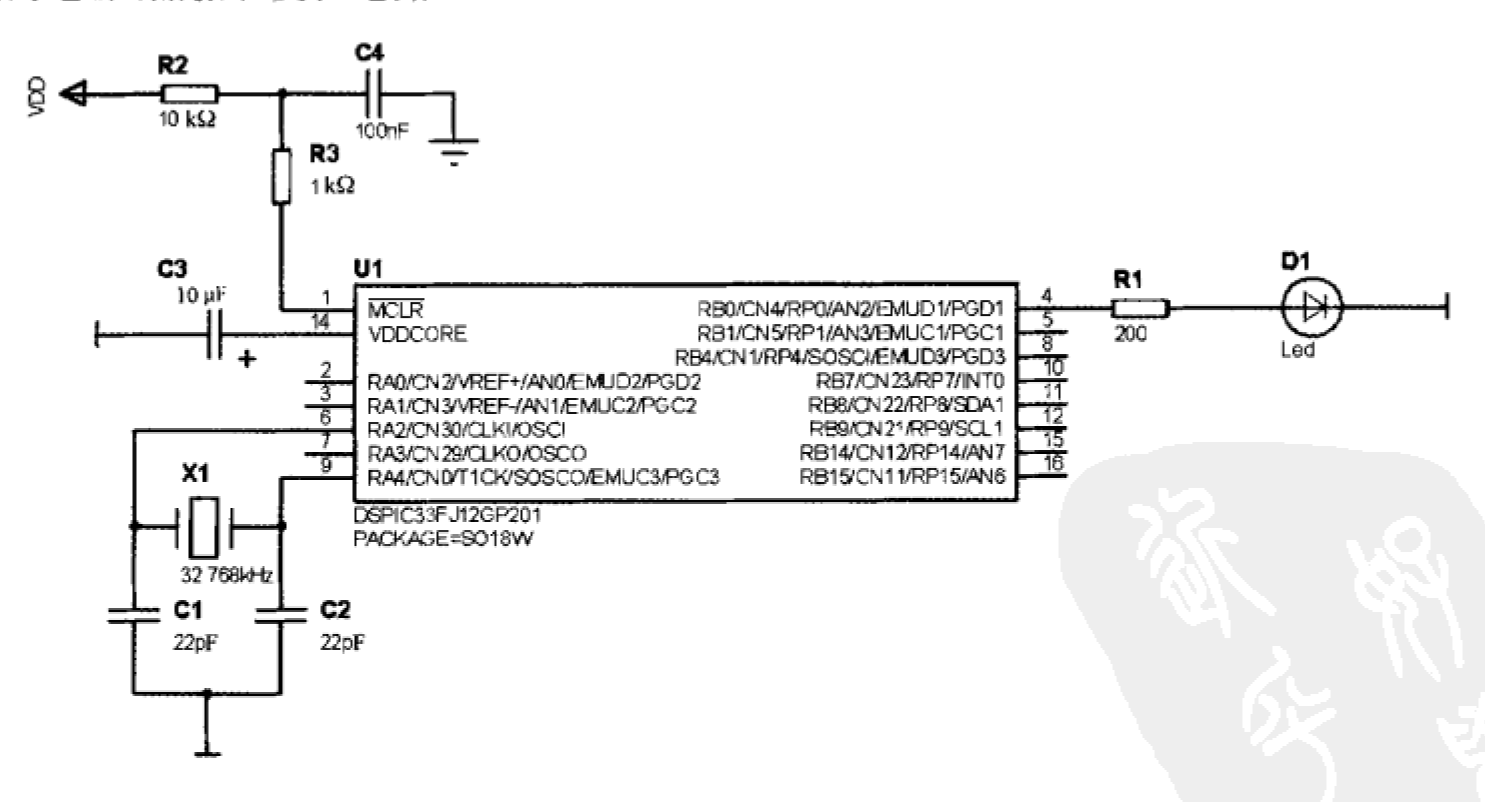
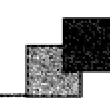


图 7.1.17 Proteus 原理图

Proteus 中，该元件封装将 13、17、18 引脚隐藏了，因此未显示出 VDD，VSS。双击 dsPIC33FJ12GP201 芯片，打开其属性对话框，如图 7.1.18 所示。



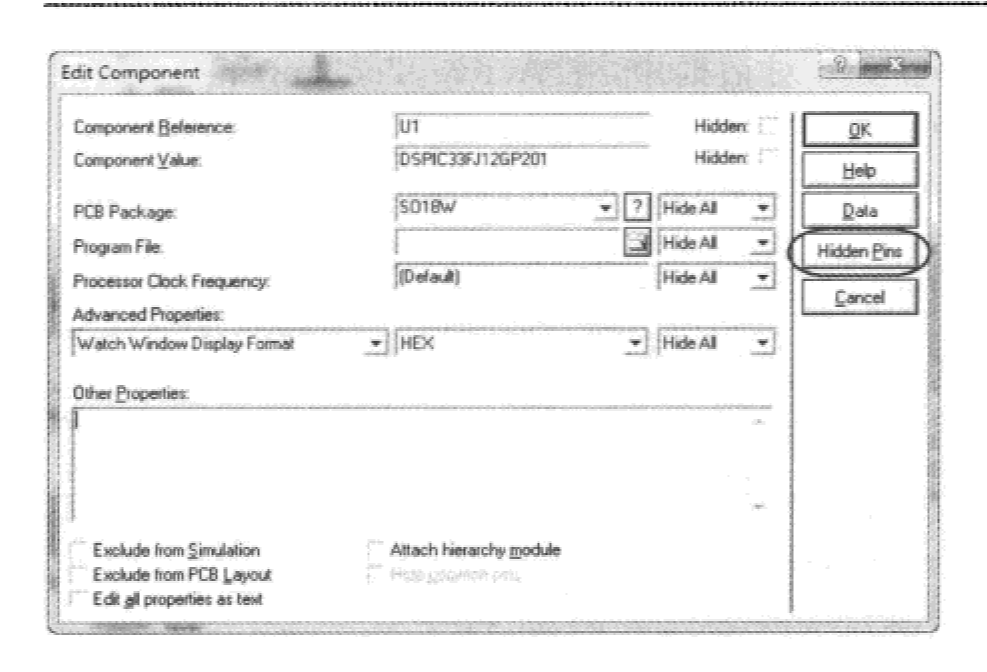


图 7.1.18　芯片属性设置界面

单击对话框右侧的 Hidden Pins 即可查看隐藏引脚的连接情况。由图 7.1.19 可知，隐藏引脚已经与 VDD，VSS 相连。

2. 建立 MPLAB 工程

打开 MPLAB IDE，选择菜单栏上的 Project→Project Wizard，在弹出的工程向导提示下，可以快速新建一个 MPLAB 工程，如图 7.1.20 所示。

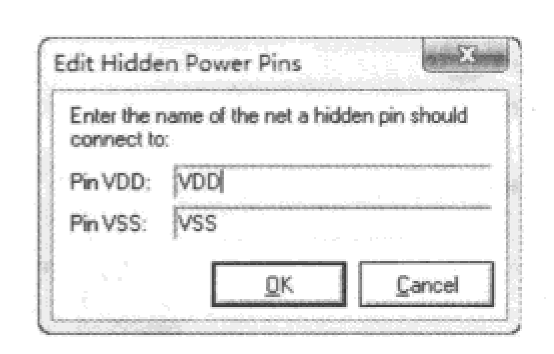


图 7.1.19　隐藏管脚页面

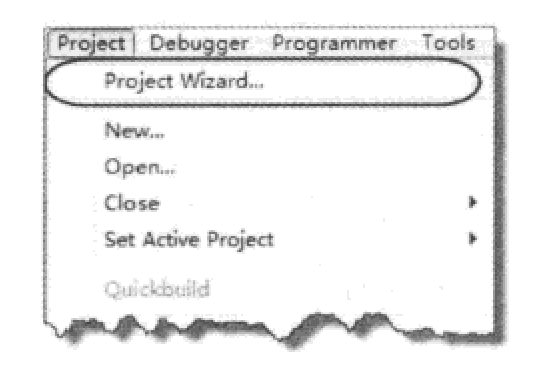


图 7.1.20　选择工程向导

在工程向导的 Step one 对话框中指定芯片 dsPIC33FJ12GP201，如图 7.1.21 所示。

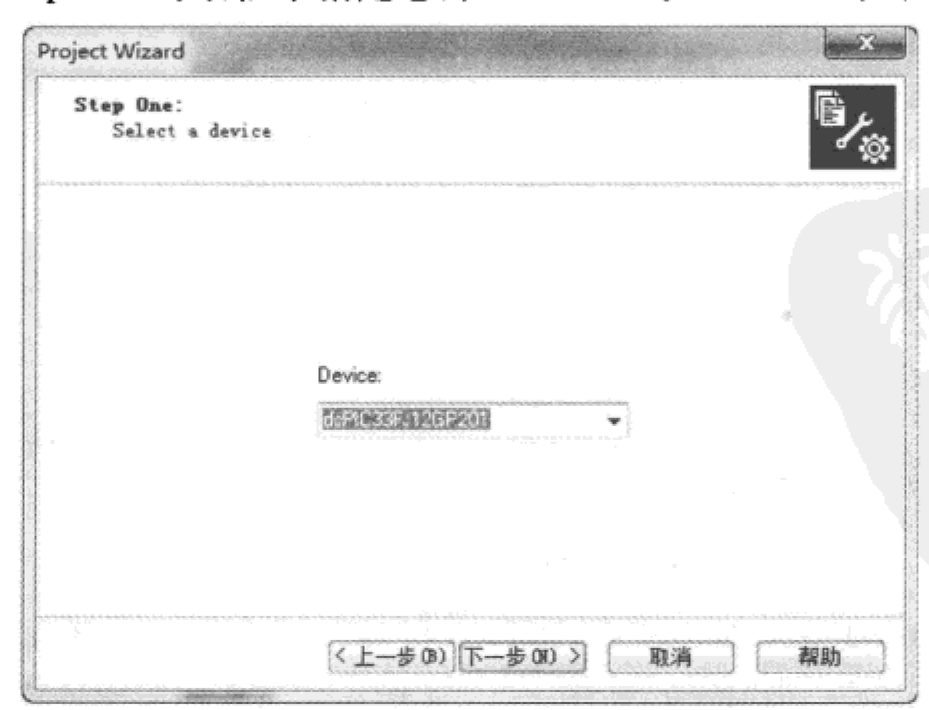


图 7.1.21　指定芯片

在工程向导的 Step two 对话框中，在 Active Toolsuit 下拉菜单中选择之前安装过的 MPLAB C30 Toolsuit。然后需要在 Location 中为 Toolsuit 的每一个组件指定路径，不然 MPLAB IDE 无法定位到这些组件，如图 7.1.22 所示。

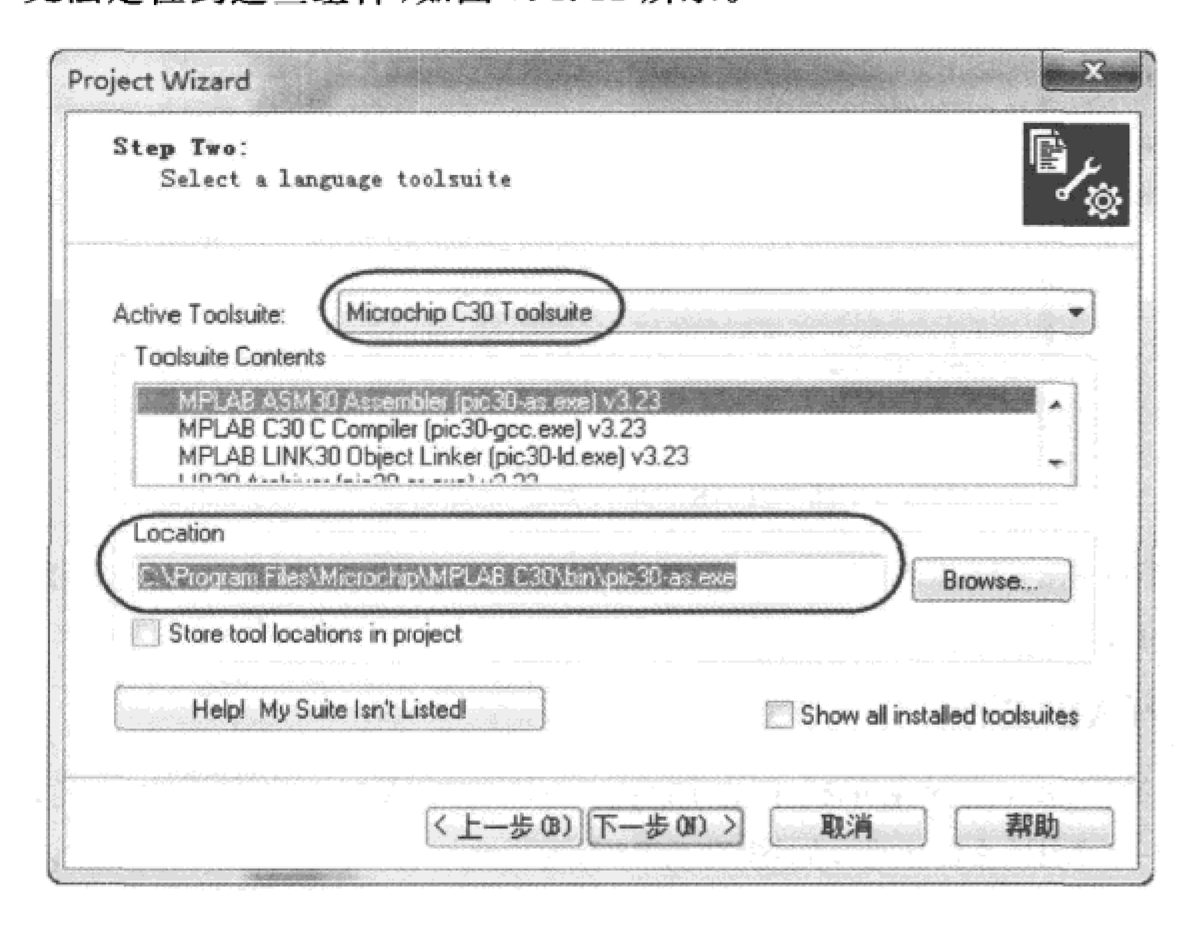


图 7.1.22　选择工具

在工程向导的 Step Three 对话框中，确定工程的名称和保存路径，如图 7.1.23 所示。

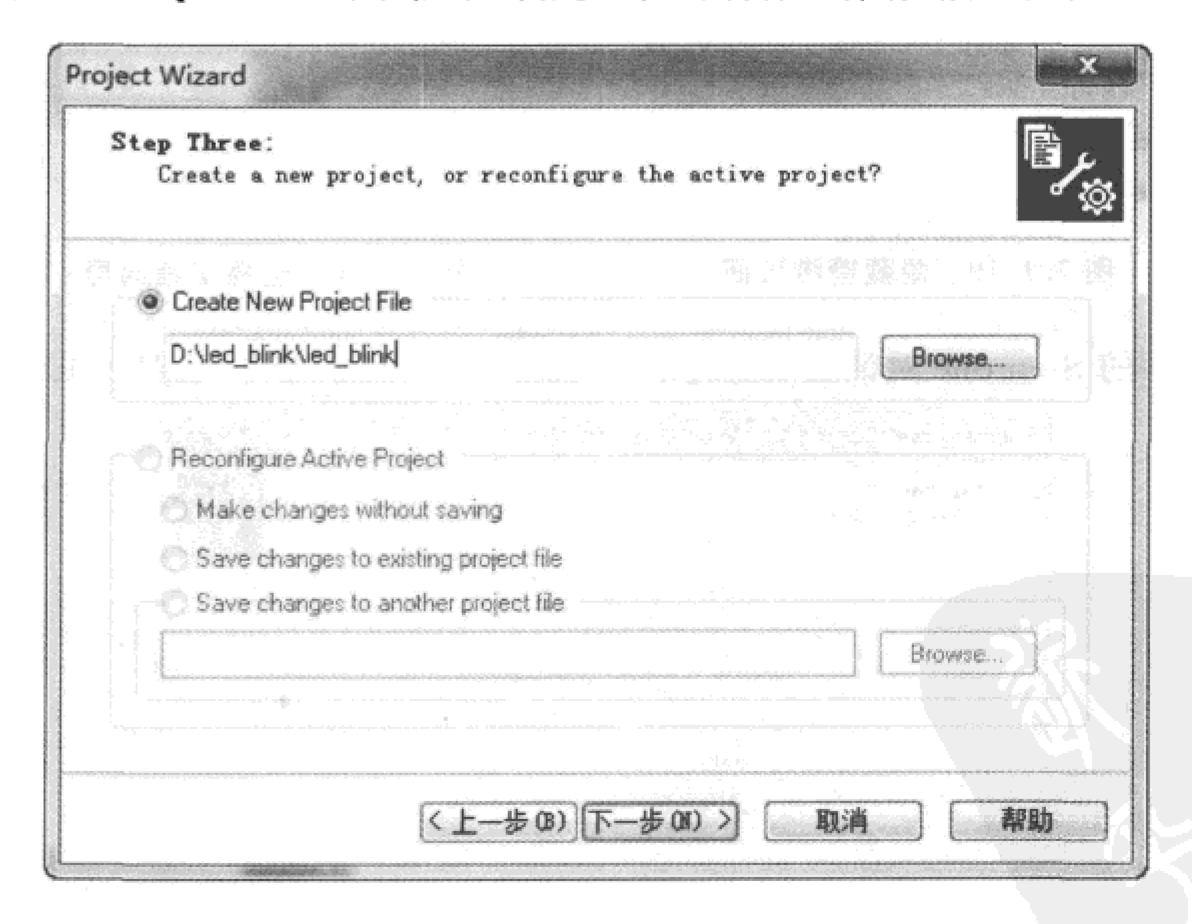
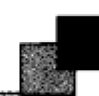


图 7.1.23　确定名称及路径

在工程向导的 Step Three 对话框中，可以直接将已存在的文件添加到工程中，如果没有需要添加的文件，单击“下一步”按钮，如图 7.1.24 所示。



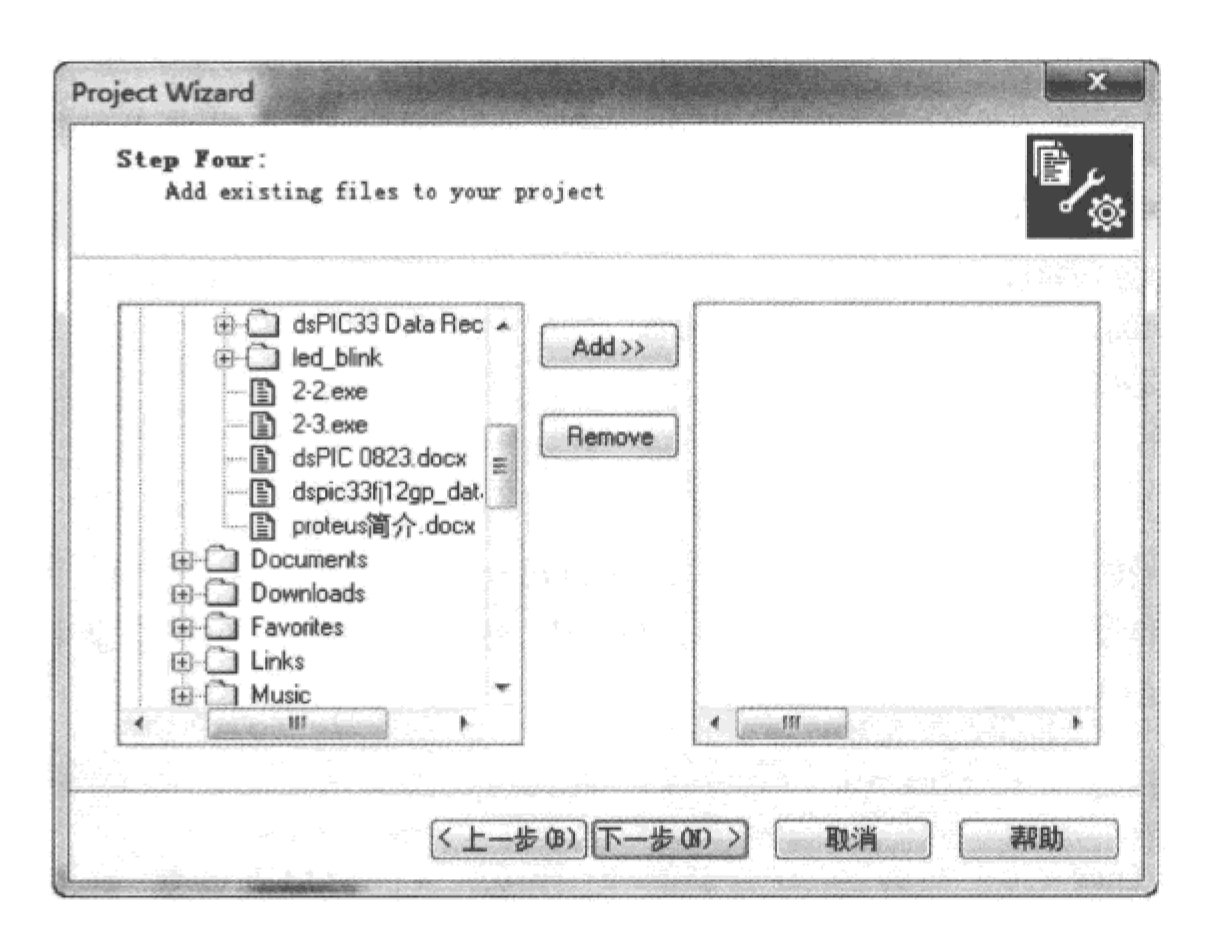


图 7.1.24　向工程添加文件

工程建立后会同时生成工作空间，保存到同一目录下即可，如图 7.1.25 所示。

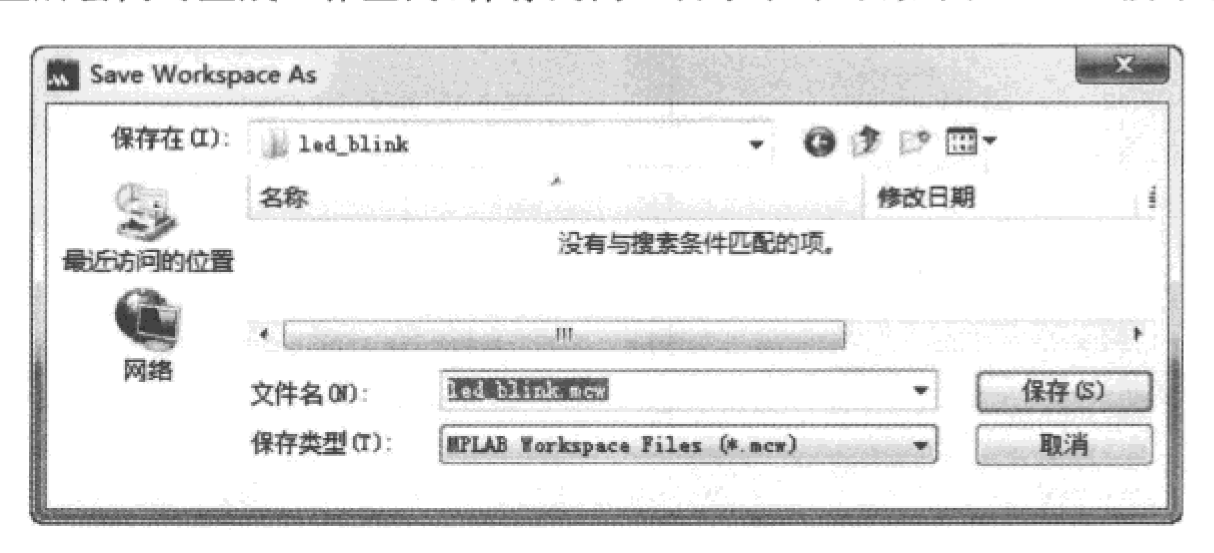


图 7.1.25　保存工作空间

编写闪烁灯程序。

单击工具栏按钮，在编辑窗口中编写以下程序：

```
#include<p33FJ12GP201.h>
#define uint unsigned int
void delay(uint a)                    //延时程序
{
    Uintx,y;
    for(x=a;x>0;x--)
      for(y=100;y>0;y--);
}
void main()
{
```



```
        TRISBbits.TRISB0 = 0;          //设置 RB0 口为输出
        while(1)
        {
            PORTBbits.RB0 = 1;         //点亮 LED
            delay(1000);
            PORTBbits.RB0 = 0;         //熄灭 LED
            delay(1000);
        }
    }
```

单击工具栏按钮保存文件为.c 格式。在弹出对话框中勾选 Add File To Project 复选框并保存，直接将文件添加到工程的 Source File 中，然后向 Header File 目录中添加芯片头文件 p33FJ12GP201.h，如图 7.1.26 所示。

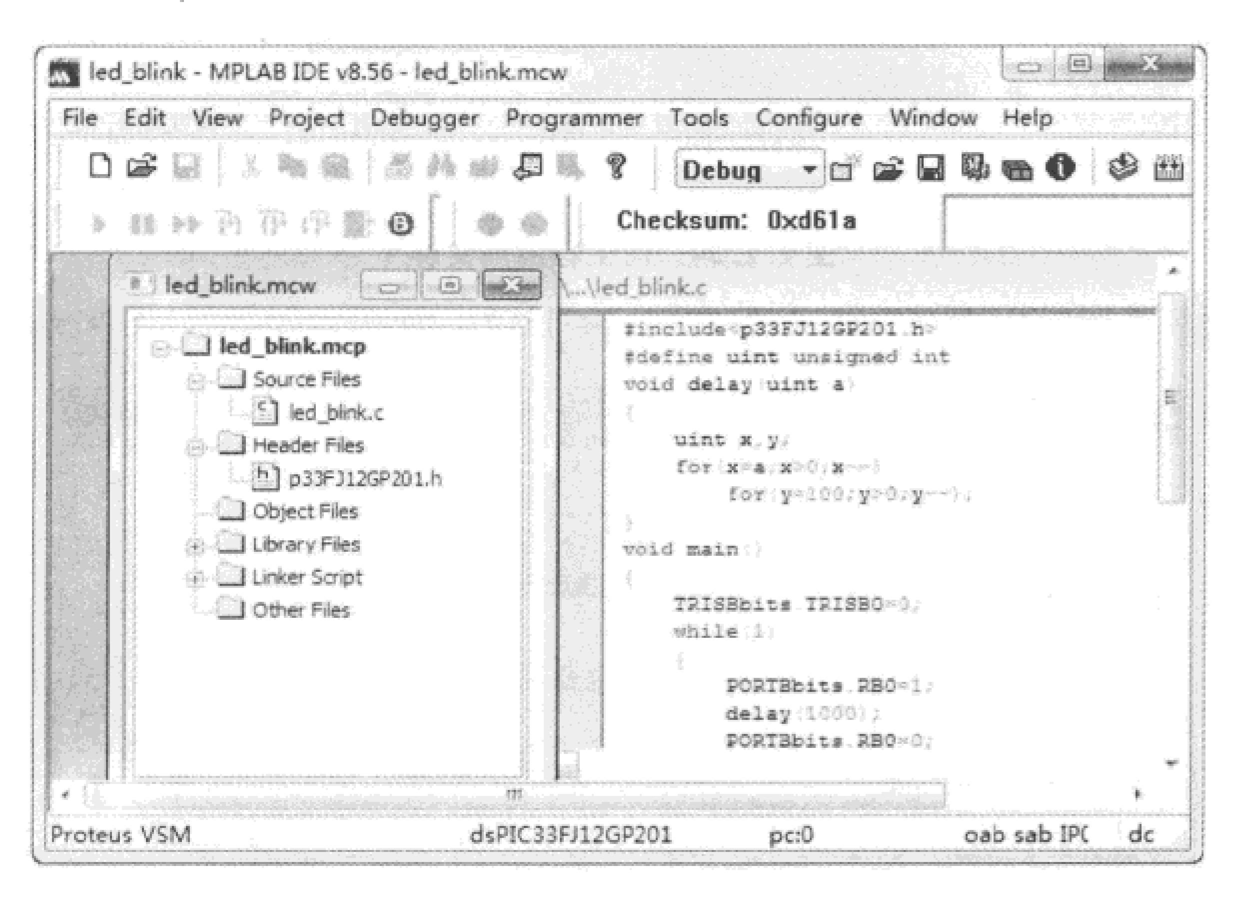


图 7.1.26　MPLAB 工程

为了提高 dsPIC 芯片的灵活性和可靠性，Microchip 公司为其设计了一些特殊功能，如看门狗、代码保护、JTAG 边界扫描接口、在线串行编程和在线仿真。为了控制芯片和使用这些特殊功能，需要用户对其配置位进行适当调整。

在菜单栏中选择 Configure→Configuration Bits… 命令，打开配置位对话框，如图 7.1.27 所示。

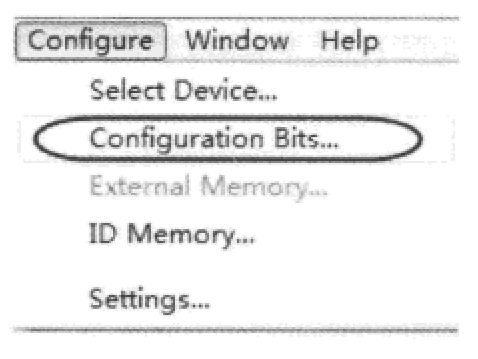
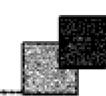


图 7.1.27　配置位

在配置位对话框中取消勾选 Configuration Bits set in code 复选框，然后用户就可以在此处修改配置位了。如果用户对配置位比较熟悉，则应选择 Configuration Bits set in code 命令，并直接在代码中使用_Config()进行修改。本例并未使用到芯片的特殊功能，使用默认设置即可。关于配置位的介绍可参考芯片的数据手册，如图 7.1.28 所示。



Configuration Bits

Configuration Bits set in code.

Address	Value	Field	Category	Setting
F80000	000F	BWRP	Boot Segment Wr	Boot Segment may be written
		BSS	Boot Segment Pr	No Boot Segment
F80004	0007	GWRP	General Code Se	General Segment may be written
		GSS	General Segment	User program memory is not code protected
F80006	0002	FNOSC	Oscillator Mode	Primary oscillator (XT, HS, EC)
		IESO	Internal Extern	Start-up device with user-selected oscillato
F80008	00C6	POSCMD	Primary Oscilla	HS Crystal Oscillator mode
		OSCIOFNC	OSCO/OSC2 Pin F	OSCO/OSC2 is clock output
		IOL1WAY	Peripheral Pin	Allow Multiple Re-configurations
		FCKSM	Clock Switching	Sw Disabled, Mon Disabled
F8000A	00DF	WDTPOST	Watchdog Timer	1:32,768
		WDTPRE	WDT Prescaler	1:128
		WINDIS	Watchdog Timer	Watchdog Timer in Non-Window mode
		FWDTEN	Watchdog Timer	Watchdog Timer always enabled
F8000C	00F7	FPWRT	POR Timer Value	128ms
		ALTI2C	Alternate I2C	I2C mapped to SDA1/SCL1 pins
F8000E	00C3	ICS	Comm Channel Se	Use PGC1/EMUC1 and PGD1/EMUD1
		JTAGEN	JTAG Port Enabl	Disabled

图 7.1.28　配置位设置页面

单击工具栏按钮编译程序，输出如下信息，如图 7.1.29 所示。

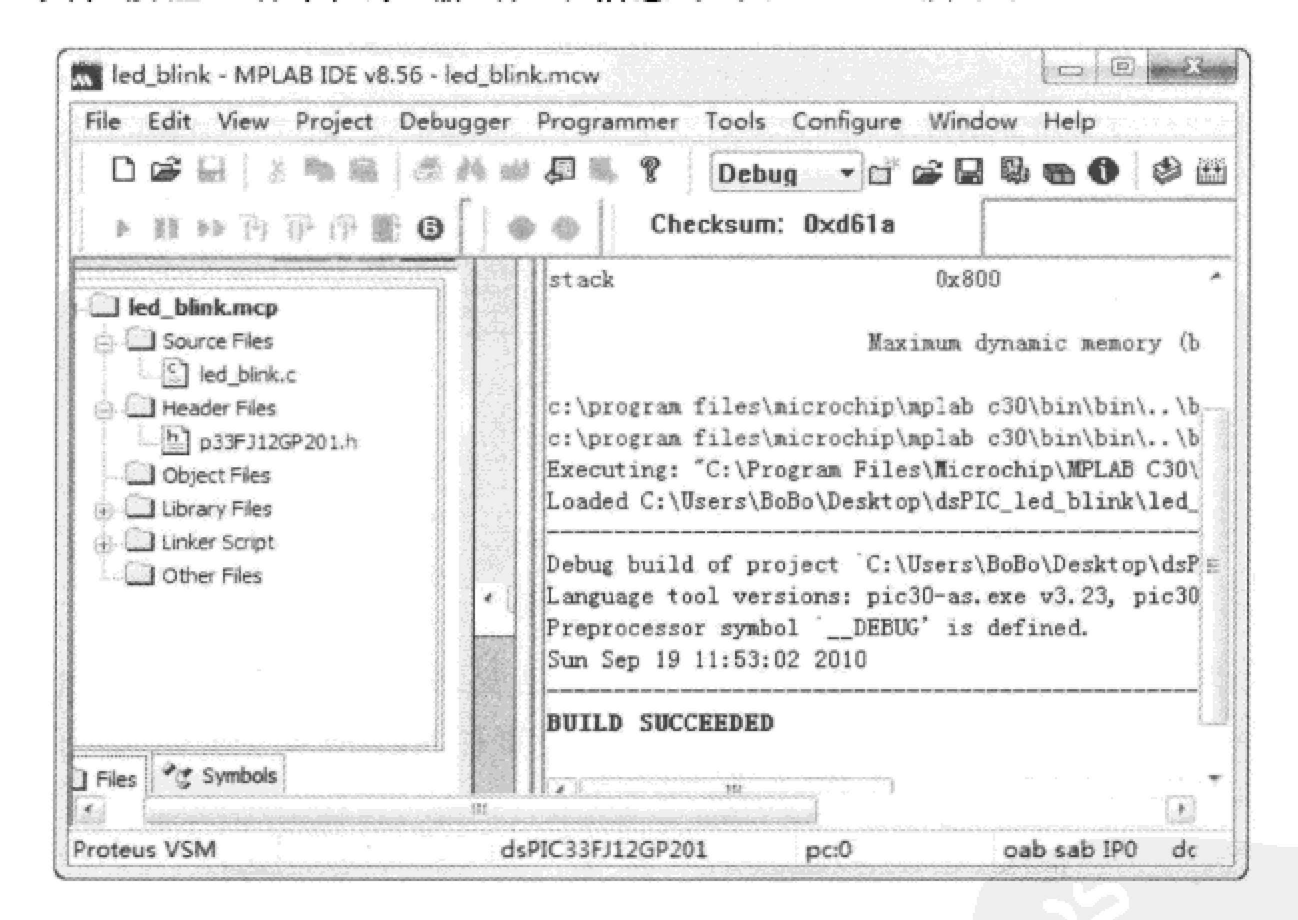


图 7.1.29　编译 MPLAB 工程

3. 调　试

选择 MPLAB IDE 菜单栏的 Debugger→Select Tool→Proteus VSM，在 MPLAB IDE 中打开 Proteus 界面。在 Proteus 界面中单击按钮打开之前绘制的原理图，如图 7.1.30 所示。

单击工具栏右侧的按钮，或选择菜单栏的 Debugger→Start Simulation 开始仿真。这时调试工具栏处于激活状态。单击按钮，进入全速运行状态，可以看到二极管不停地闪烁，如图 7.1.31 所示。

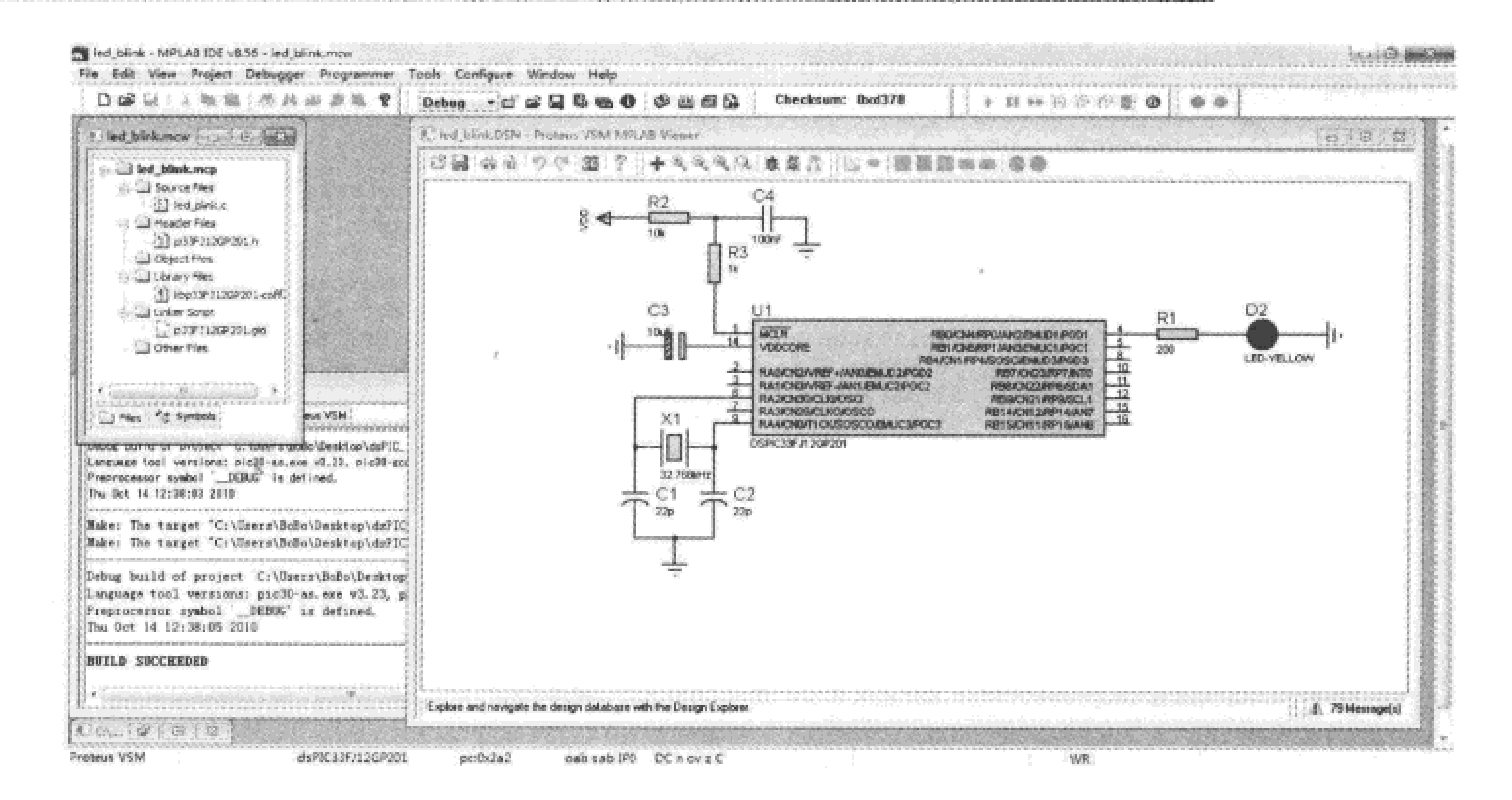


图 7.1.30 使用 Proteus VSM 调试工具

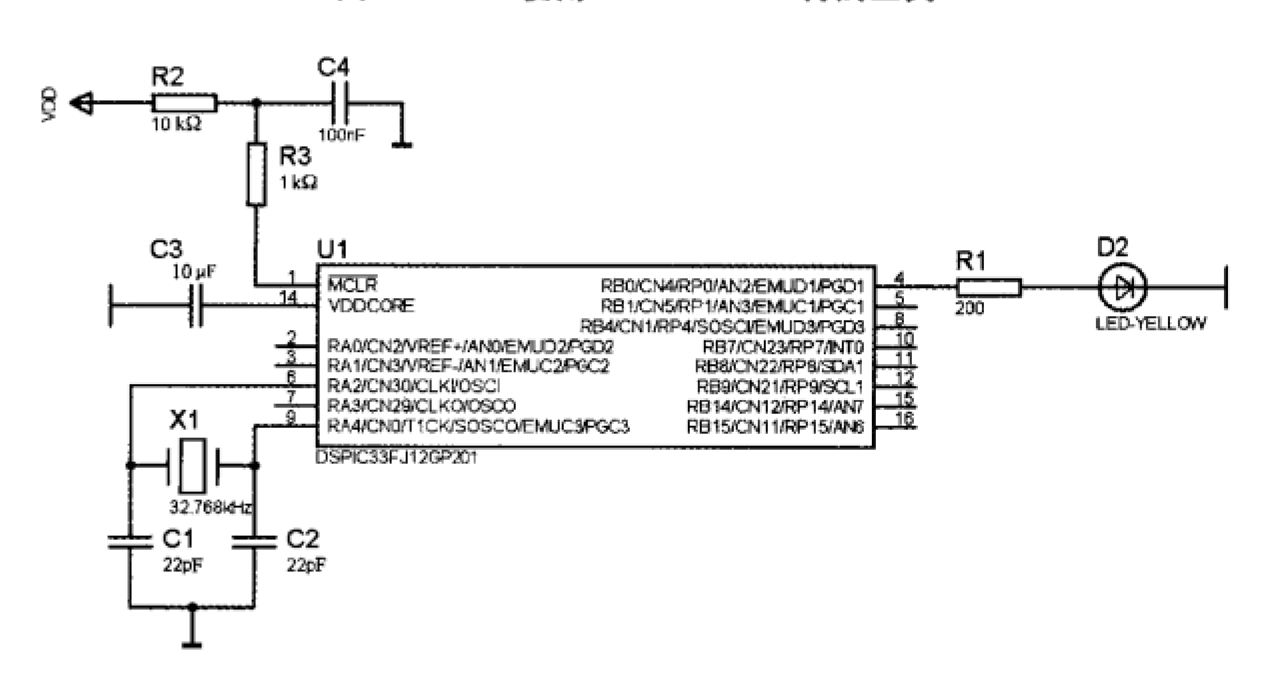


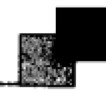
图 7.1.31 LED 灯闪烁

除此之外，MPLAB 还提供了其他观察芯片内部数据变化的工具，例如在菜单中选择 view→Special Function Registers，可以实时观察芯片的特殊功能寄存器的值，发生变化的寄存器以红色标出，如图 7.1.32 所示。

Special Function Registers

Address	SFR Name	Hex
226	U1RXREG	x0000
228	U1BRG	x0000
240	SPI1STAT	x0000
242	SPI1CON1	x0000
244	SPI1CON2	x0000
248	SPI1BUF	x0000
2C0	TRISA	x001F
2C2	PORTA	x0000
2C4	LATA	x0000
2C6	ODCA	x0000
2C8	TRISB	xC392
2CA	PORTB	x0000
2CC	LATB	x0000

图 7.1.32 特殊功能寄存器



若选择 view→Disassenbly Listing 选项，可以看到程序的反汇编指令，并以箭头标注下一步执行指令的位置，如图 7.1.33 所示。

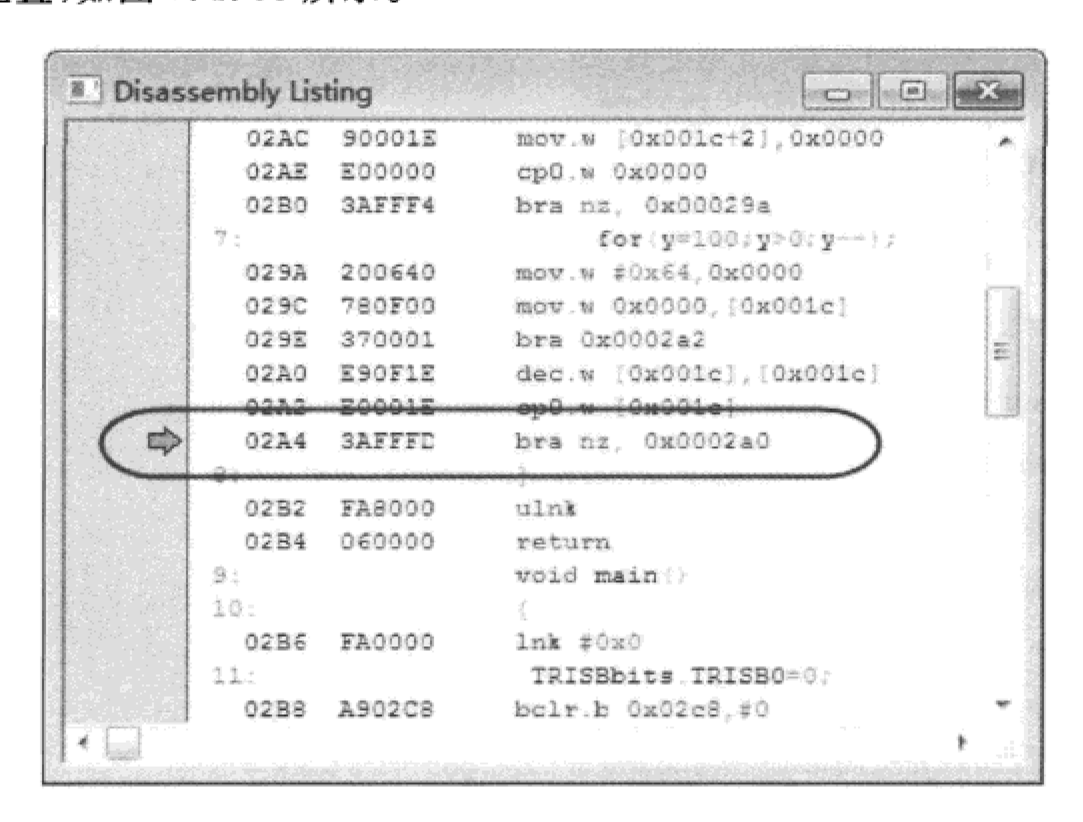


图 7.1.33　反汇编指令列表

7.1.3　dsPIC 外围驱动模块简介

MATLAB/Simulink device blocksets 与 dsPIC 芯片的外围驱动模块一一对应，用户可以通过这些模块创建模型，直接由概念生成可执行的代码。

Embedded Target for the Microchip dsPIC DSC 集成了 MATLAB/Simulink 和 MPLAB IDE 工具，通过 RTW 生成的 C 代码与 MPLAB IDE 达到 Simulink 模型的基于 C 的嵌入式实时实现。

模块简介如下：

(1) ADC Config：模数转换配置模块。该模块可设置模拟量的输入端口、输出数字量的格式、转换结果寄存器、A/D 控制寄存器等，驱动芯片的 A/D 设备正常工作。

(2) Write Port Output：写入输出端口模块。该模块可指定将数据写入到哪一个端口，即能按位指定，也能按字节指定端口。

(3) dsPIC33FXX Main：dsPIC33FXX 芯片配置模块。每个基于 dsPIC 芯片的模型都需要使用该模块。它和芯片的配置位相对应，例如设置晶振、计时器等。

(4) Port Config：端口配置模块。该模块指定端口的输入/输出状态，既能按位指定，也能按字节指定。

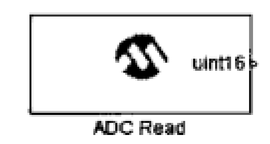
在 MATLAB/Simulink device blocksets 的安装目录下有更加详细的帮助文件。如果用户安装 MPLAB IDE 和 MATLAB/Simulink device blocksets 时选择了默认目录，则可在 C:\Program Files\Microchip\MPLAB IDE\Tools\MATLAB\dsPIC_Matlab\doc\help 目录下找到每一个模块的 HTML 格式的帮助文件，例如打开文件 dsPIC ADC Read. html，则出现图 7.1.34所示的帮助信息。

ADC Read for dsPIC Devices

ADC Read

Read from specified buffer of ADC peripheral.

Block



Description

Configure any ADC of a dsPIC device to read the data available at ADC read buffer.

Ports

Input

None

Output

Data of the type uint16 will be available at output port of the block.

图 7.1.34　ADC Read 模块帮助信息

7.2　dsPIC 外围驱动模块应用

7.2.1　数模转换实验

ADC Read 模型实现了控制 dsPIC33FJ12GP202 的片内 ADC，将模拟电压输入转换为数字量输出的功能。本模型用到了 dsPIC Configuration 模块库中的 dsPIC33fXX Main、ADC Config 和 Port Config；dsPIC Run Time Library 模块库中的 ADC Read 和 Write Port Output 模块。建立图 7.2.1 所示的模型，并将其命名为 adc. mdl。

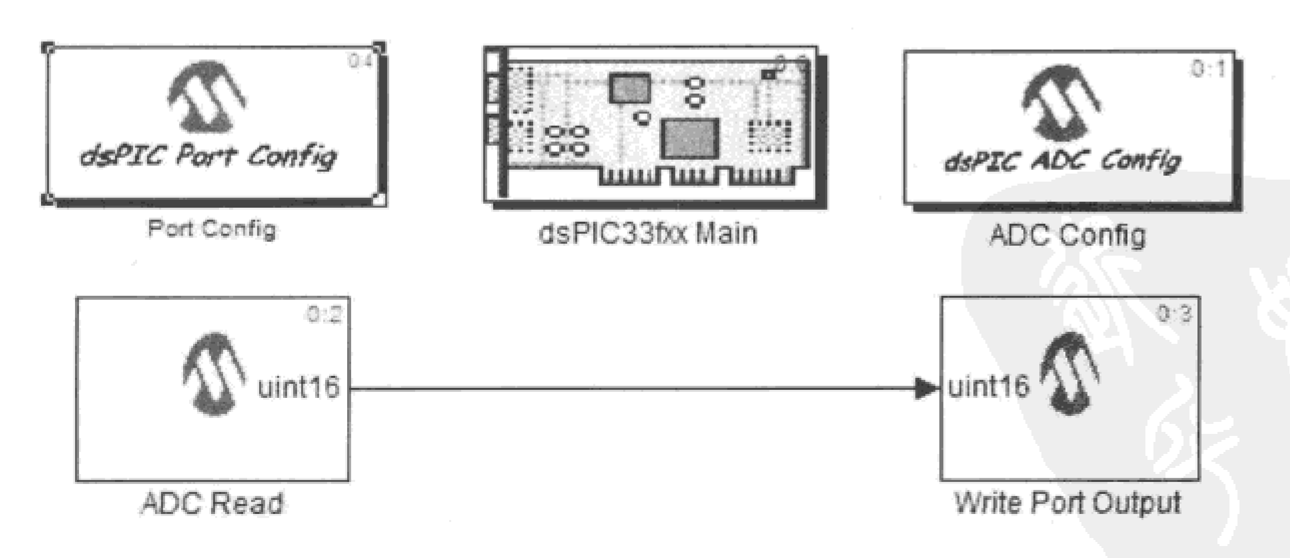
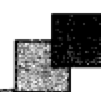


图 7.2.1　ADC 模型



1. 模块设置

双击 ADC Read 模块，打开设置界面。选择片内 1 号 ADC，由 buffer0 读出数据，输出数据类型为 uint16，如图 7.2.2 所示。

双击 Write Output Port 模块，打开设置界面。选择 B 端口，并具体指定输出到 B 端口的 0～15 号引脚，如图 7.2.3 所示。

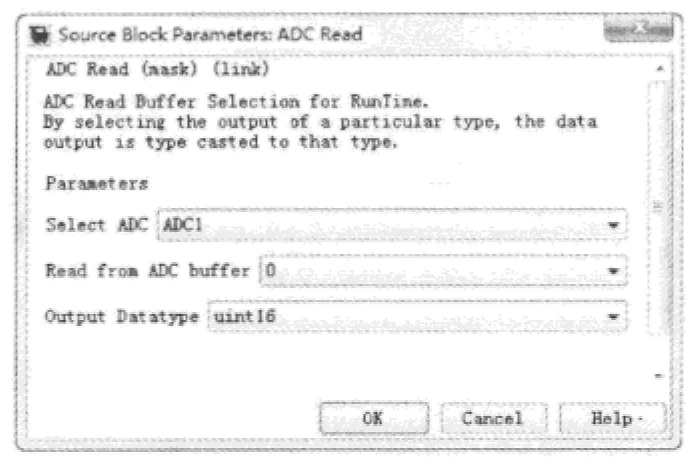



图 7.2.2　ADC Read 模块设置

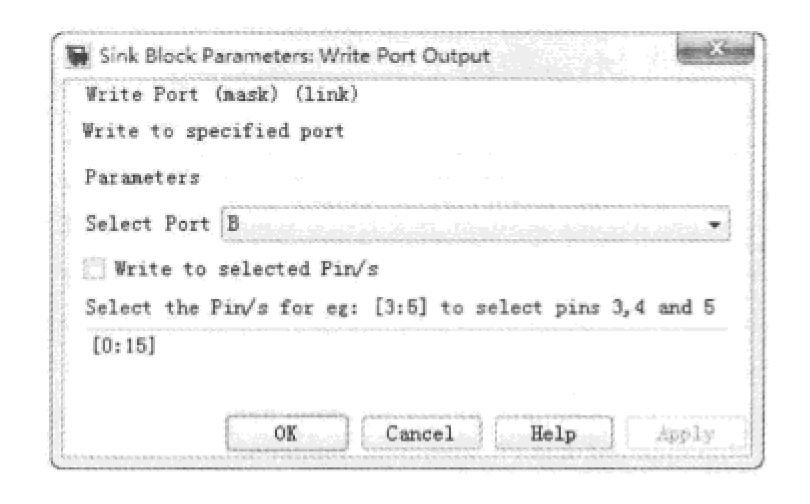


图 7.2.3　Write Output Port 模块设置

dsPIC33fXX Main、ADC Config 和 Port Config 模块用于设置芯片的配置位和特殊功能寄存器，以确定芯片的工作方式。如果用户对 dsPIC 芯片比较熟悉，设置这些模块并不困难。

dsPIC33FXX Main 模块对应于芯片的配置位，双击打开其设置界面。在 Oscillator Configuration 选项卡，选择芯片型号为 dsPIC33FJ12GP202，计时器选择 Timer1。

振荡器选择快速 *RC* 振荡器(Fast RC oscillator)。由于未选用主振荡器，其模式设置为禁用(Primary Disabled)。

看门狗和代码保护功能本例并未用到，因此 Watchdog Configuration 和 Code Protect Configuration 选项卡采用默认设置即可(默认为 disable 状态)，如图 7.2.4 所示。

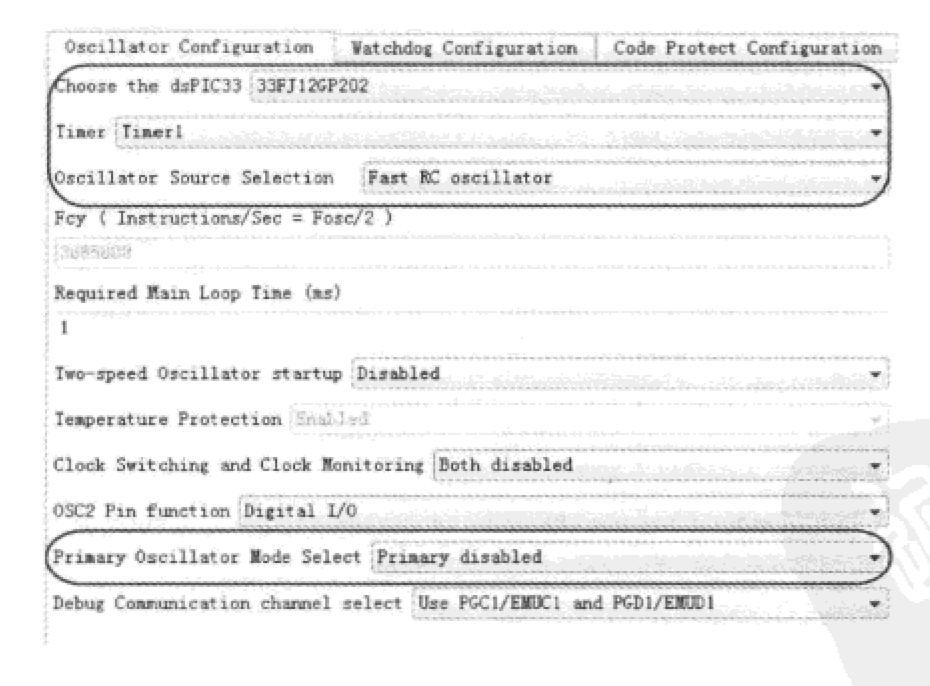


图 7.2.4　dsPIC33FXX Main 模块设置

ADC Config 对应于芯片的 ADxCONx 控制寄存器，双击打开其设置界面。

在 ADC 选择中选用 1 号 ADC；数据输出格式为整数右对齐(Integer(Dout＝0000 dddd dddd dddd))；

采样时钟源选择内部时钟源，时钟源结束采样后自动启动转换(Internal counter ends

sampling and starts convertion(auto-convert))；

由于选用单通道采样，因此同步采样选择位无效；使能 ADC 采样自动启动(ADC Sample Auto-Start)；

转换的参考电压分别设为 VDD 和 VSS(ADREF+=Avdd；ADREF-=Avss)；

转换模式选择 10 位 A/D(10-bit 4-channel ADC operation)；

将 AN1 设置为模拟输入通道，并启用输入扫描(Scan Input Selections for CH0+ during Sample A bit)，如图 7.2.5 所示。

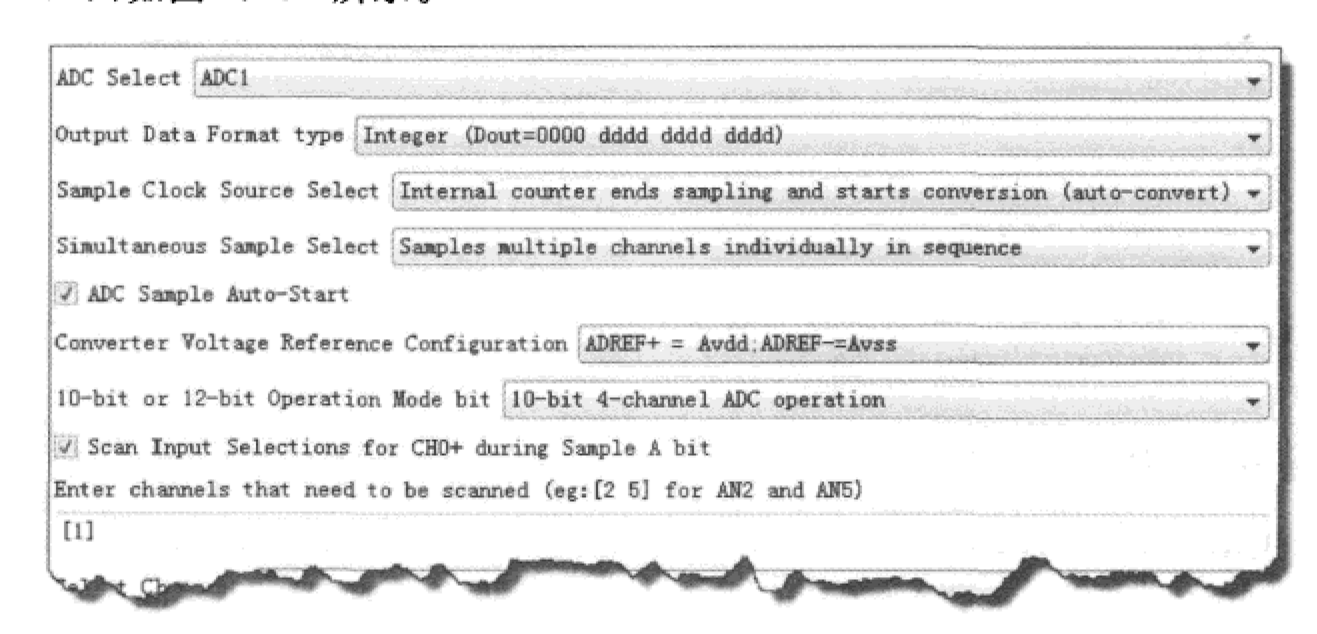


图 7.2.5　ADC Config 模块设置

在通道选择中选择转换 CH0 通道(Converts CH0)。

在交替输入选择中选择只作为采样 A 的输入(use channel input for Sample A)。

将 AN1 设为通道 CH0 采样 A 的同向输入(AN1)。

由于未选择 CH1、CH2、CH3 及采样 B，其他三项同向输入设置位无效。

每完成一次采样-转换就产生一次中断，读出转换结果(SMPI 设为 1)，如图 7.2.6 所示。

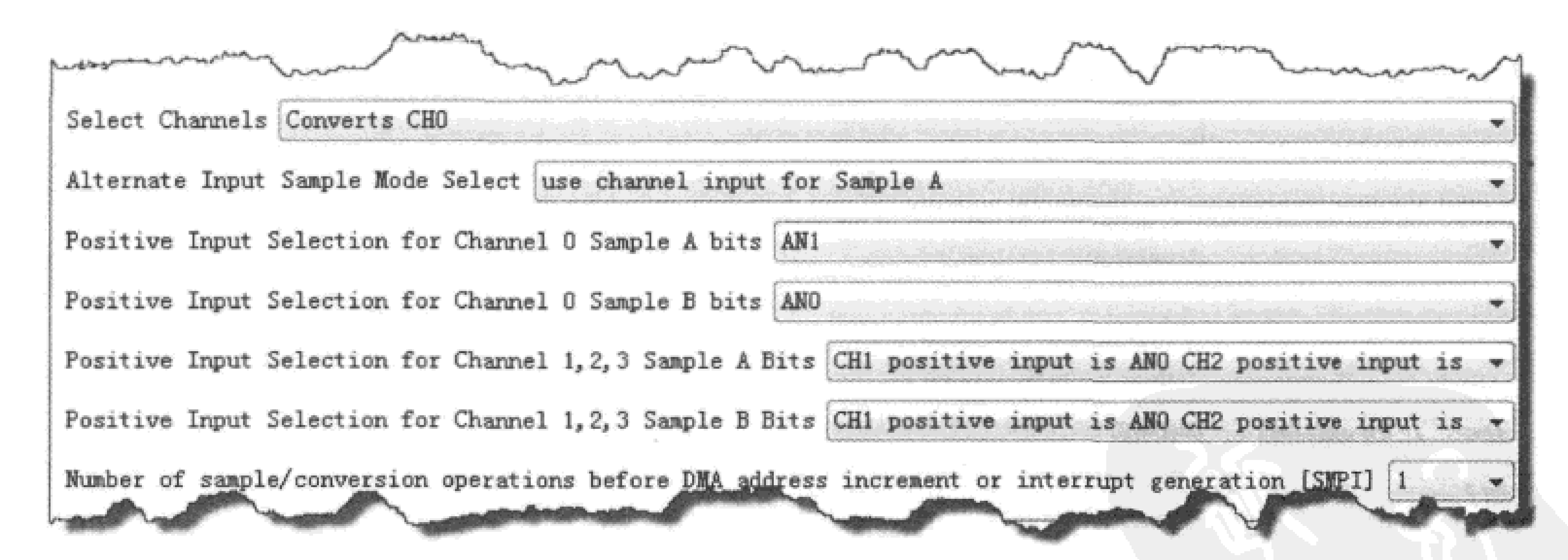
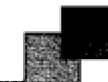


图 7.2.6　ADC Config 模块设置

在 A/D 转换时钟源中选择系统时钟(Clock derived from system clock)。

自动采样时间设置为 12 个 A/D 转换周期(12Tad)，其实只要保证其不小于 A/D 转换所需的最短时间要求即可。

芯片数据手册中介绍，完成一次 10 位 A/D 转换需要 12 个 A/D 转换周期，因此 A/D 转换时间设置为 12 个 A/D 转换周期。



使能中断后，会在稍后的生成代码中出现一个中断服务程序框架，用户可以手动添加所需代码，本例中不需要使用，如图7.2.7所示。

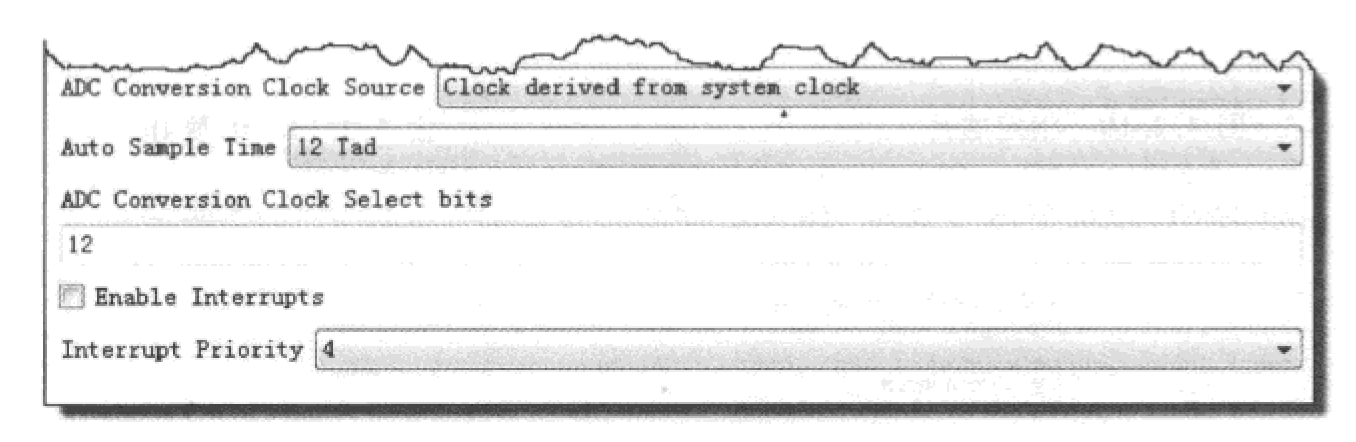


图7.2.7 ADC Config 模块设置

Port Config 模块对应于特殊功能位 TRISx，双击打开其设置界面。选择端口B，并将其0～15引脚设置为输出状态，如图7.2.8所示。

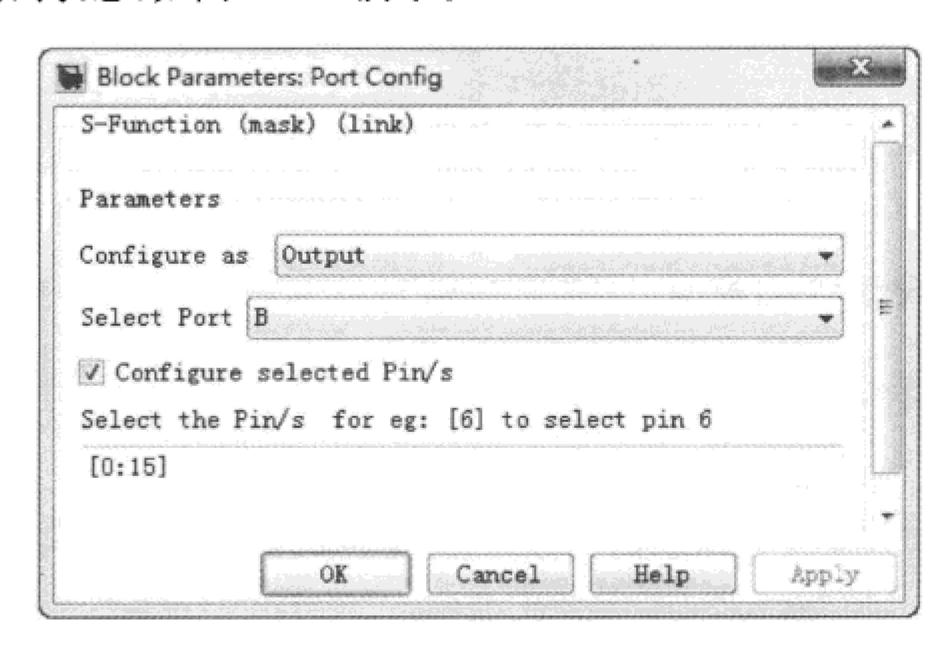


图7.2.8 Port Config 模块设置

2. 模型参数设置

在模型窗口的菜单栏选择 Simulation→Configuration Parameters，设置模型参数。在 Solver 界面，设置仿真截止时间为 inf，设置求解器为定步长离散求解器，步长可根据需要另行指定，或使用 auto，如图7.2.9所示。

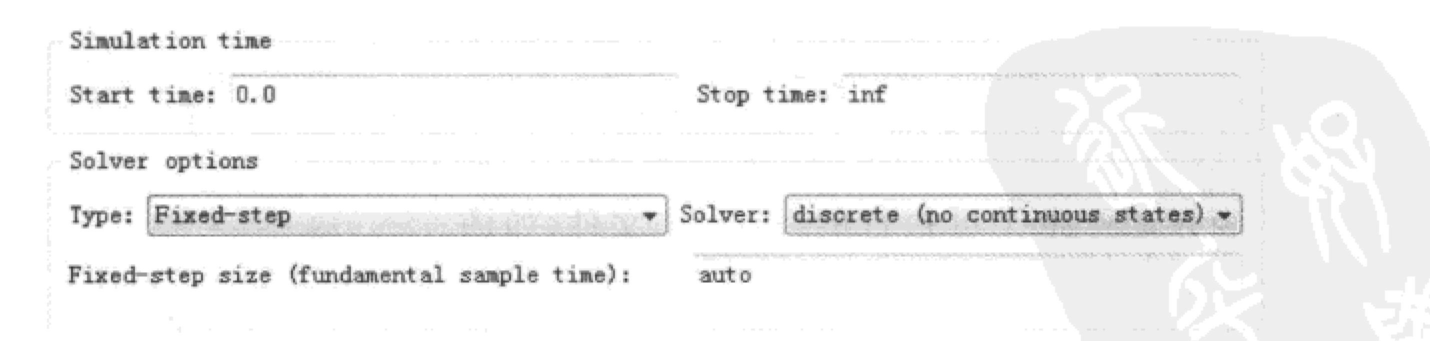


图7.2.9 设置仿真时间，求解器

在 Hardware Implimentation 界面，设置器件类型为 dsPIC，如图7.2.10所示。

在 Real-Time Workshop 界面，设置 TLC 文件为 dsPIC_stf.tlc，如图7.2.11所示。

Embedded hardware (simulation and code generation)
Device vendor: Microchip
Device type: dsPIC

图 7.2.10　指定芯片

Target selection
System target file: dsPIC_stf.tlc　Browse...
Language: C

图 7.2.11　设置 tlc

在 Report 界面，勾选所有复选框，便于后期检查及跟踪，如图 7.2.12 所示。

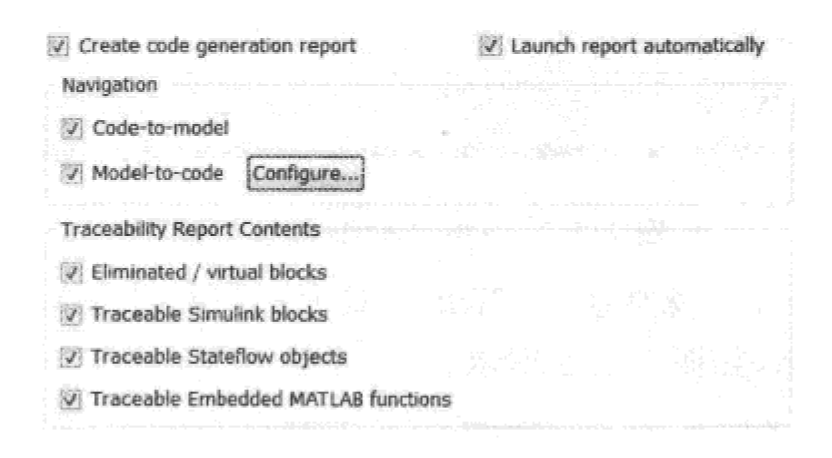


图 7.2.12　报告界面设置

3. 自动生成代码

完成这一系列的设置，单击模型工具栏的按钮，即生成代码，报告如图 7.2.13 所示。

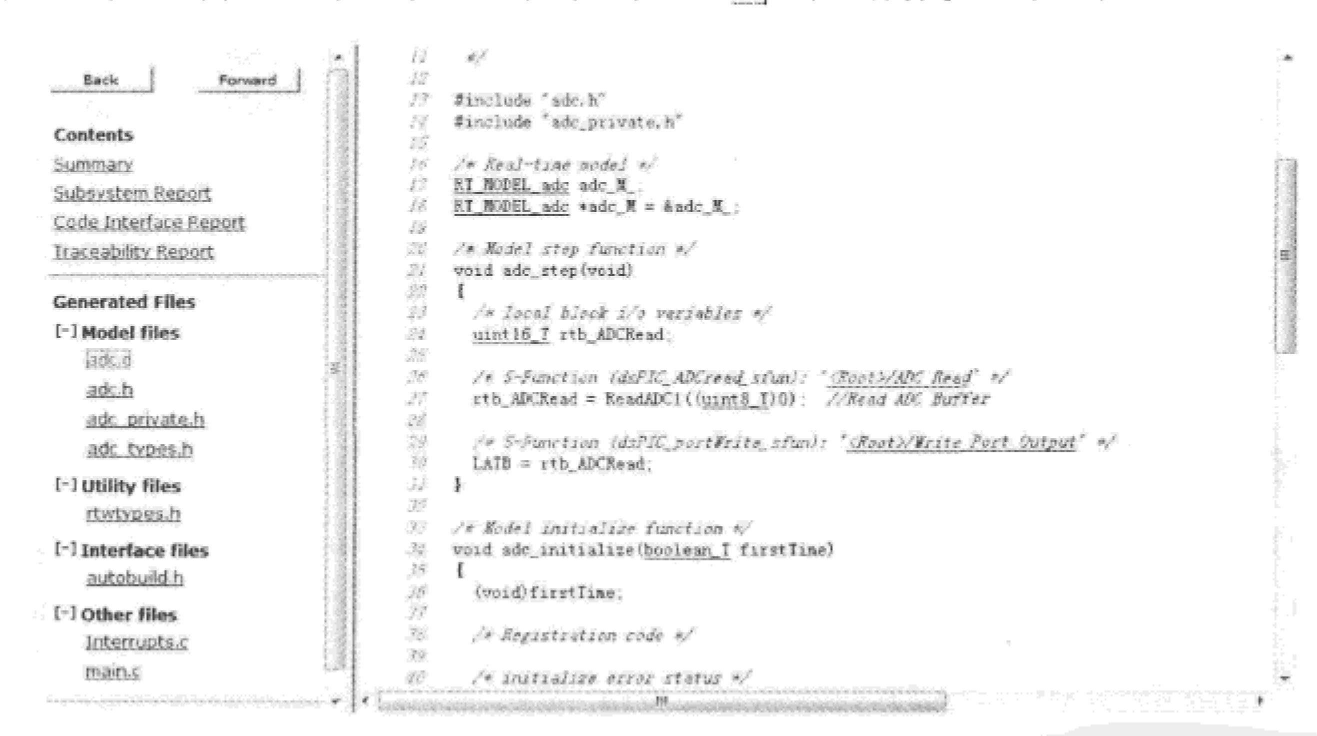


图 7.2.13　代码生成报告

代码位于 MATLAB 当前目录下的 adc_dspic_ert\src 文件夹，并且已经自动产生了可执行的 .hex、cof 文件，如图 7.2.14所示。

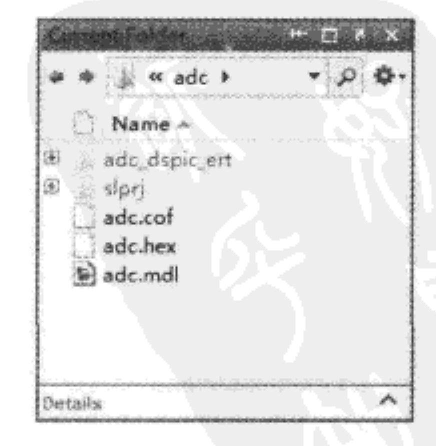
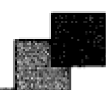


图 7.2.14　自动生成 HEX 和 COF 文件

4. 虚拟硬件测试

在 Proteus 中建立测试原理图，选用 dsPIC33FJ12GP202 芯片，其外围电路与第 7.1.2 节中介绍的 dsPIC33FJ12GP201 基本相同，只是将晶振频率设置为 8MHz。在 AN1 端口连接



了一个电位器,可以调整输入模拟电压的大小。在 RB0～RB9 连接了发光二极管,用来显示 A/D 转换结果,如图 7.2.15 所示。

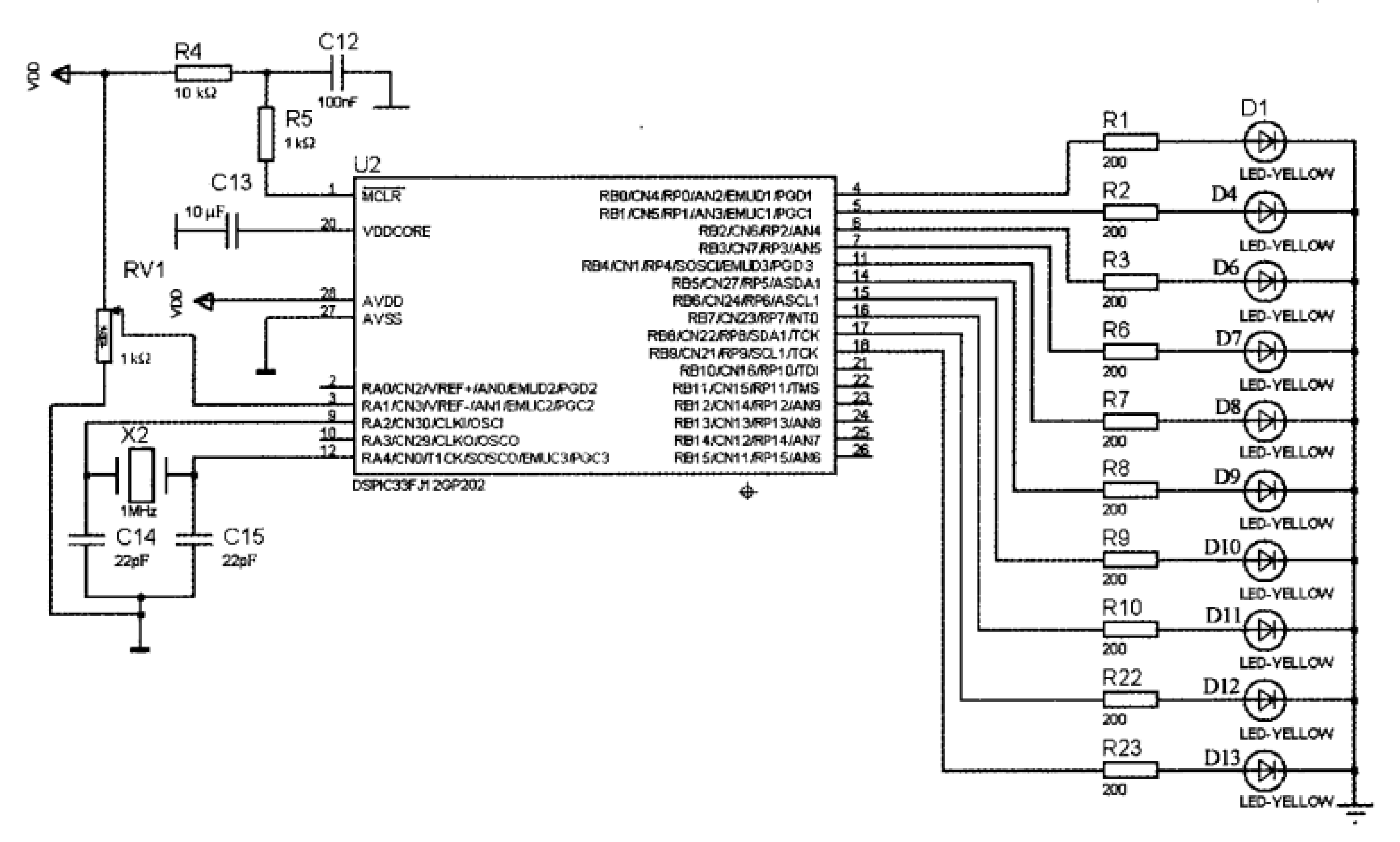


图 7.2.15 Proteus 原理图

双击芯片,打开设置界面,在 Program File 中指定生成的 adc. hex 或 adc. cof 文件,如图 7.2.16 所示。

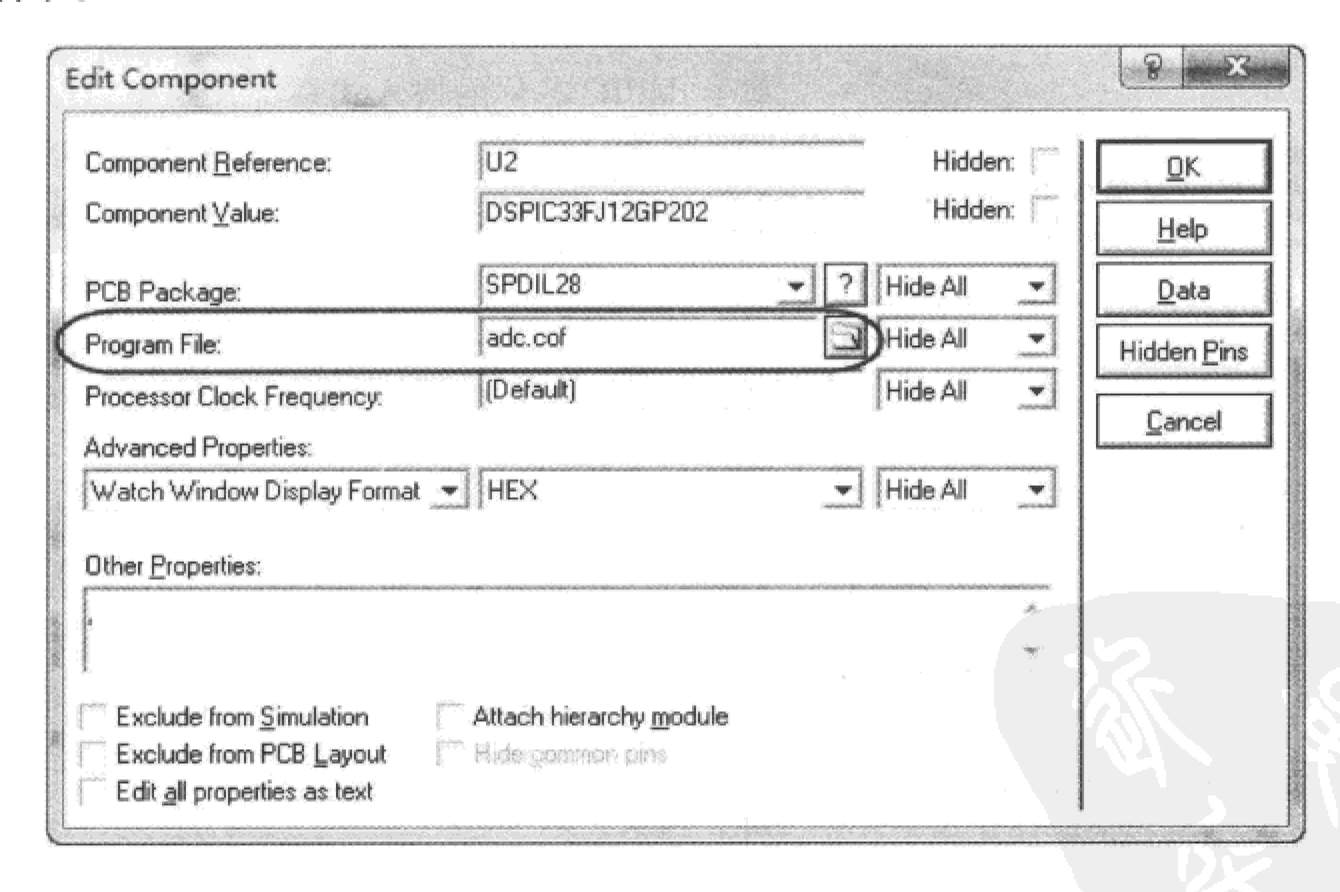


图 7.2.16 加载可执行文件

运行测试原理图。当调整电位器时,AN1 端模拟电压发生变化,右侧的发光二极管也随之亮灭变化。实验结果显示由模型自动生成的代码完全实现了模型所表达的功能,如图 7.2.17 所示。

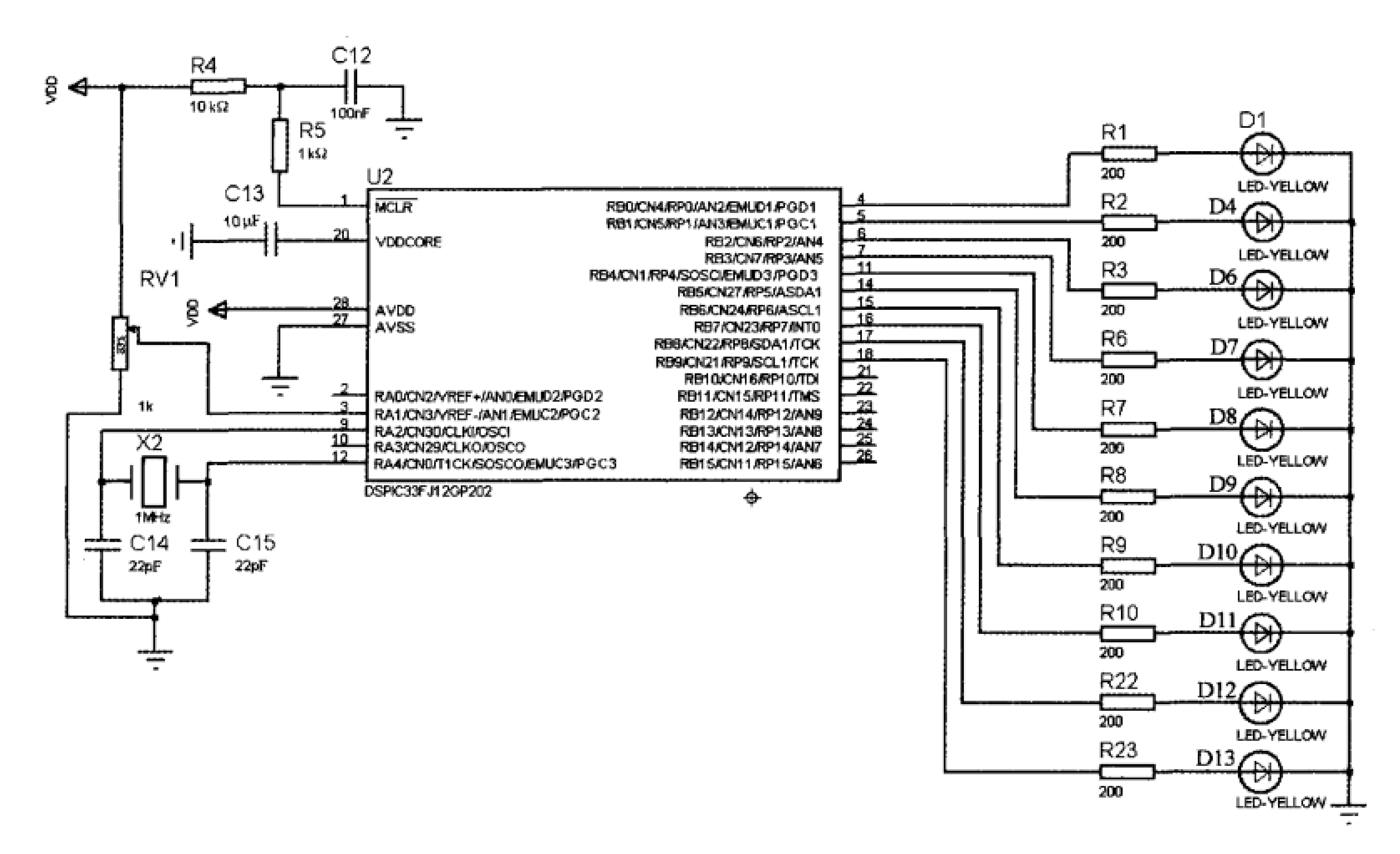


图 7.2.17 仿真结果

容易看出，在 Microchip 公司提供的 Blocksets 帮助下，配合 dsPIC_stf. tlc，用户完全可以使用由模型自动生成的代码实现嵌入式系统的开发，而不需要添加一行手写代码，这将极大地提高开基于 dsPIC 芯片的嵌入式系统发效率。

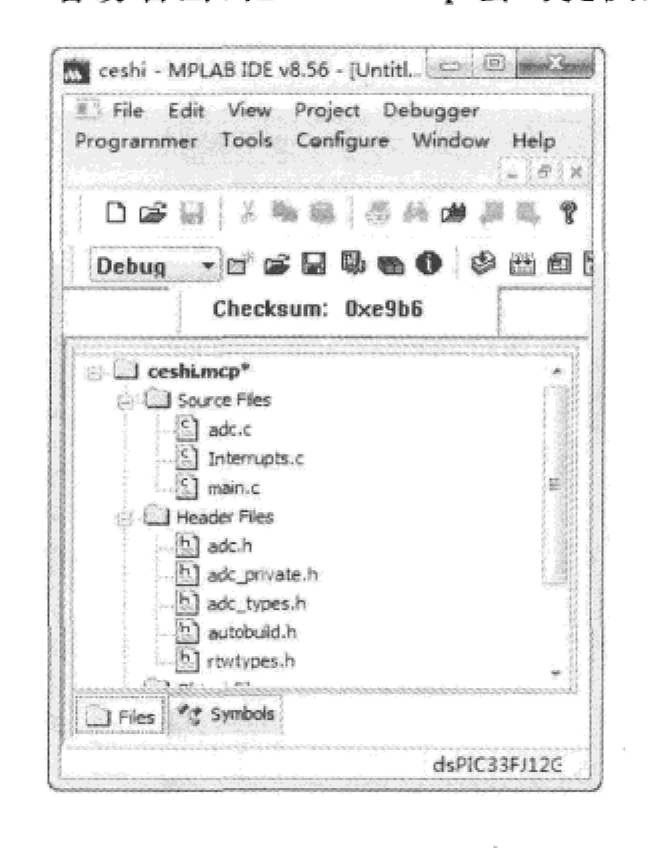


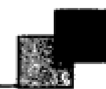
图 7.2.18 MPLAB 工程

更进一步看，10 位 ADC 输出的转换数据其实就是 0～1023 中某个值的二进制表示，这 10 位由 0、1 组成的二进制编码输出到 B 端口便表现为二极管的亮灭。为了更直观地显示测试结果，下面将采用数码管来显示转换结果。相应地，代码和测试原理图需要一定修改。

参考第 7.1.2 节内容在 MPLAB 中建立工程，将生成的代码添加到工程中，如图 7.2.18 所示。

由于数码管并不能直接将这些二进制编码转换为对应的十进制数字显示出来，因此还需要一段数码管显示程序来完成这项功能。打开 adc. c 文件，作如下修改：

```
……
#include "adc.h"
#include "adc_private.h"
/* Real-time model */
RT_MODEL_adcadc_M_;
```



```
RT_MODEL_adc * adc_M = &adc_M_;
const uint16_T table[] = {0x00FC,0x0060,0x00DA,0x00F2,0x0066, //数码管可识别的0～9编码
0x00B6,0x00BE,0x00E0,0x00FE,0x00F6};
void delay() //延时函数
{
    uint16_T i;
    for(i = 2000;i > 0;i -- );
}
void disp(uint16_T a,uint16_T b,uint16_T c,uint16_T d)  //显示函数
{
    LATB = 0x7000|table[a];//数码管显示千位数字
    delay();
    LATB = 0xB000|table[b];//数码管显示百位数字
    delay();
    LATB = 0xD000|table[c];//数码管显示十位数字
    delay();
    LATB = 0xE000|table[d];//数码管显示个位数字
    delay();
}
/ * Model step function * /
void adc_step(void)
{
  / * local block i/o variables * /
  uint16_T rtb_ADCRead,a,b,c,d;       //声明中间变量

  / * S-Function (dsPIC_ADCread_sfun): '<Root>/ADC Read' * /
rtb_ADCRead = ReadADC1((uint8_T)0);  //Read ADC Buffer
    a = rtb_ADCRead/1000;           //计算转换结果的千位数值
    b = rtb_ADCRead % 1000/100;     //计算转换结果的百位数值
    c = rtb_ADCRead % 100/10;       //计算转换结果的十位数值
    d = rtb_ADCRead % 10;           //计算转换结果的个位数值
  / * S-Function (dsPIC_portWrite_sfun): '<Root>/Write Port Output' * /
    disp(a,b,c,d);                  //调用显示函数
}
……
```

保存文件后重新编译，生成.hex文件。之后打开测试原理图，添加4位共阴极数码管，并将二极管删去，如图7.2.19所示。

通过第5.1.2节介绍的连线标签将数码管的段选端口与RB7～0连接，分别标注为a、b、…、f、g、dp。位选端口与RB15～12连接，分别标注为1、2、3、4。

打开在MPLAB中建立的工程，选择MPLAB IDE菜单栏的Debugger→Select Tool→Proteus VSM，打开修改过的原理图。单击按钮●开始调试，单击按钮▶进入全速运行状态

后，数码管上显示了模数转换的十进制结果，如图 7.2.20 所示。其表示的实际模拟电压量为 Vref * 788/1024，如果读者对用二进制数显示的电压观看不习惯，自己可以进一步用 Vref * 788/1024 公式把它变成易于观察的用十进制显示的电压值。

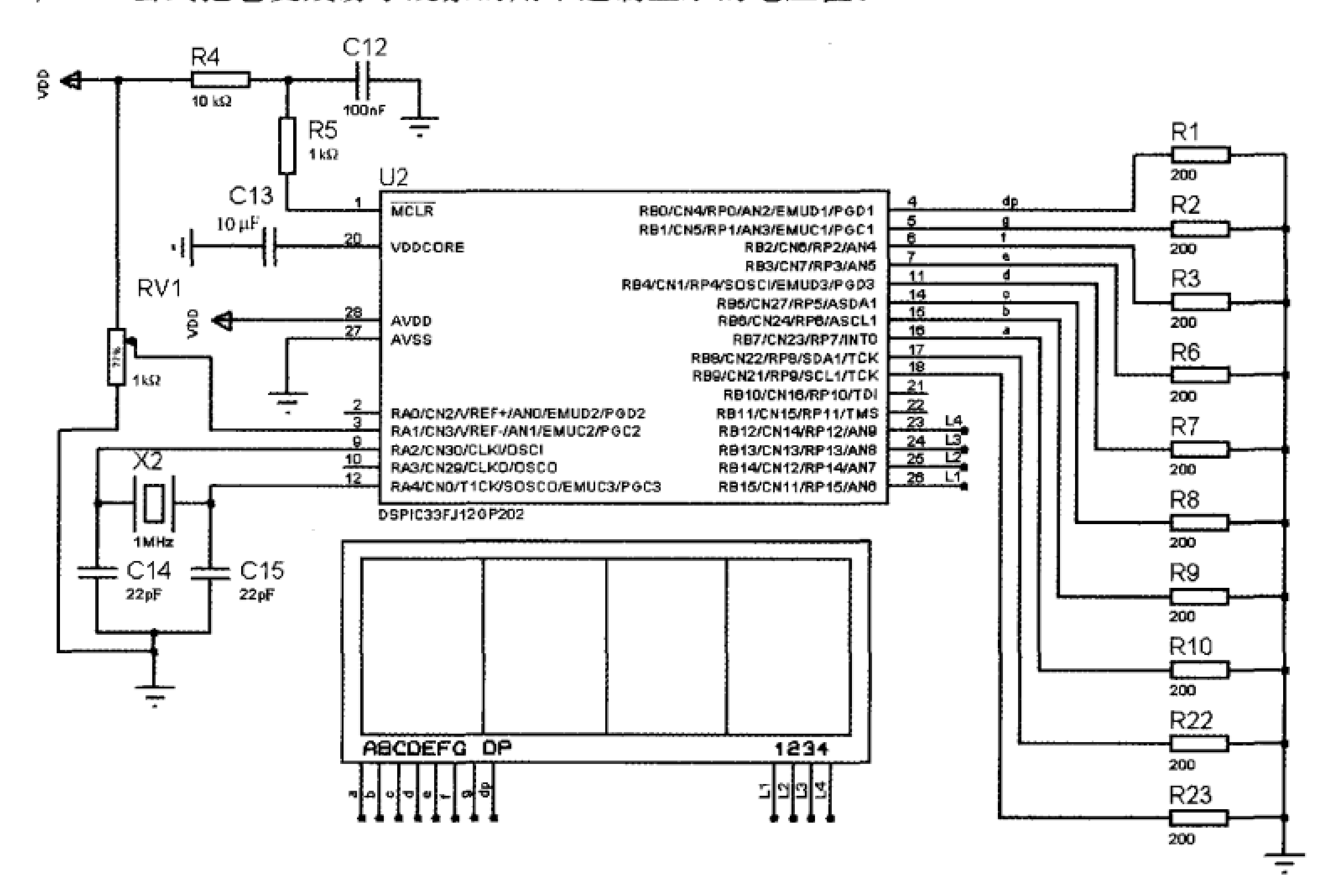


图 7.2.19　Proteus 原理图

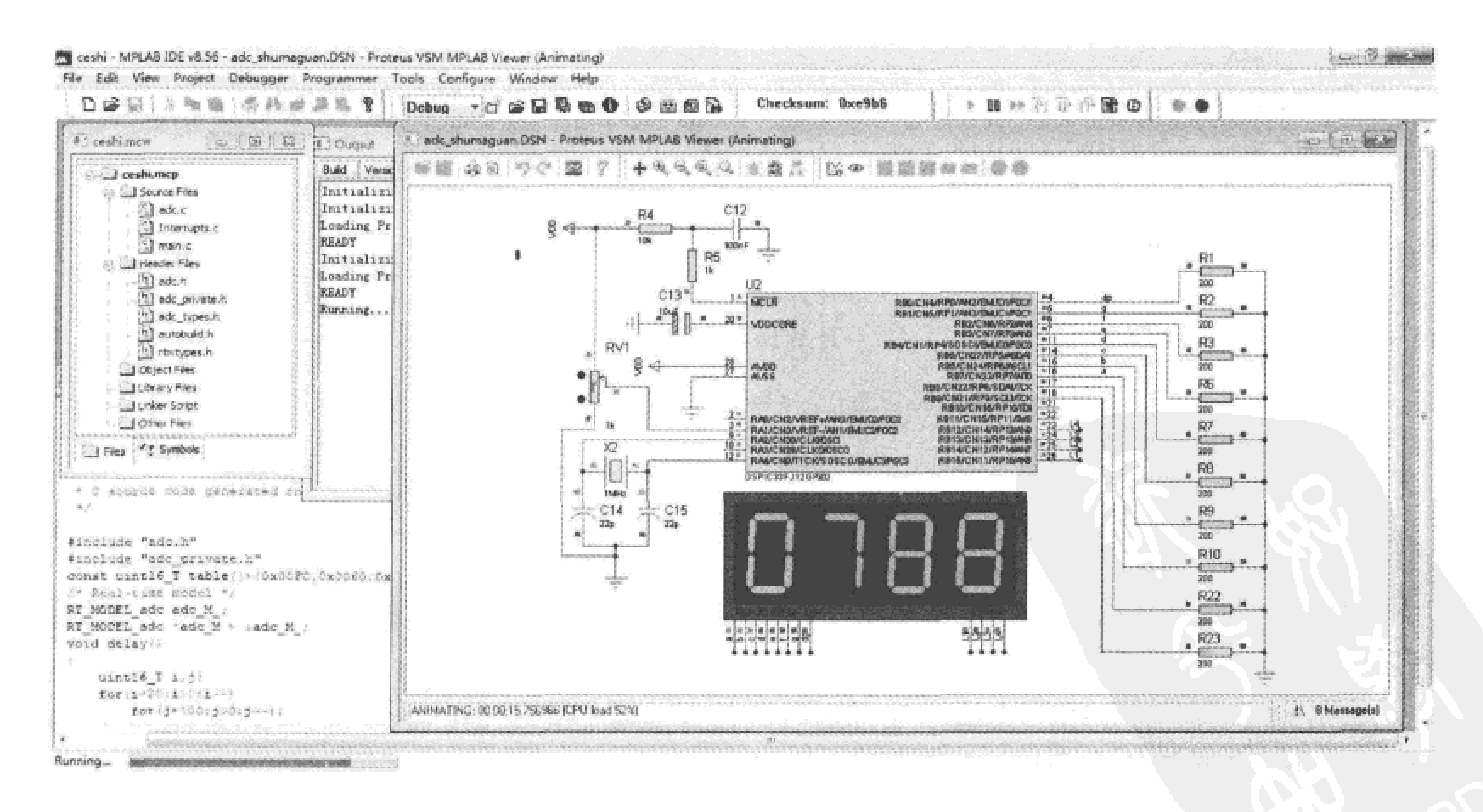
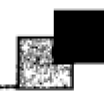


图 7.2.20　仿真结果



7.2.2　闪烁灯

1. 闪烁灯驱动模型

本例选用的处理芯片 dsPIC33FJ12GP202，支持 16 位的数据类型，可直接定义 y 的数据类型为 unit16，如图 7.2.21 所示。

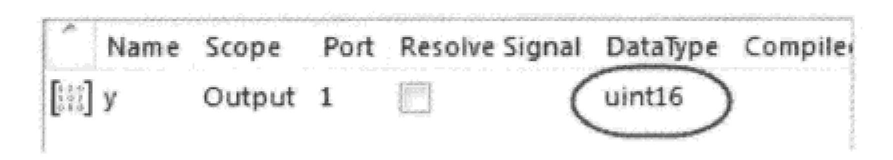


图 7.2.21　闪烁灯驱动模型数据类型

建立图 7.2.22 所示闪烁灯驱动模型，8 个状态的 y 值对应的二进制数如表 7.2.1 所示，将输出 y 与一组 LED 灯相连，即可实现 LED 灯按 y 值顺序点亮。

表 7.2.1　二进制～十进制对照

二进制	十进制	二进制	十进制
32769	1000,0000,0000,0001	2064	0000,1000,0001,0000
16386	0100,0000,0000,0010	1056	0000,0100,0010,0000
8196	0010,0000,0000,0100	576	0000,0010,0100,0000
4104	0001,0000,0000,1000	384	0000,0001,1000,0000

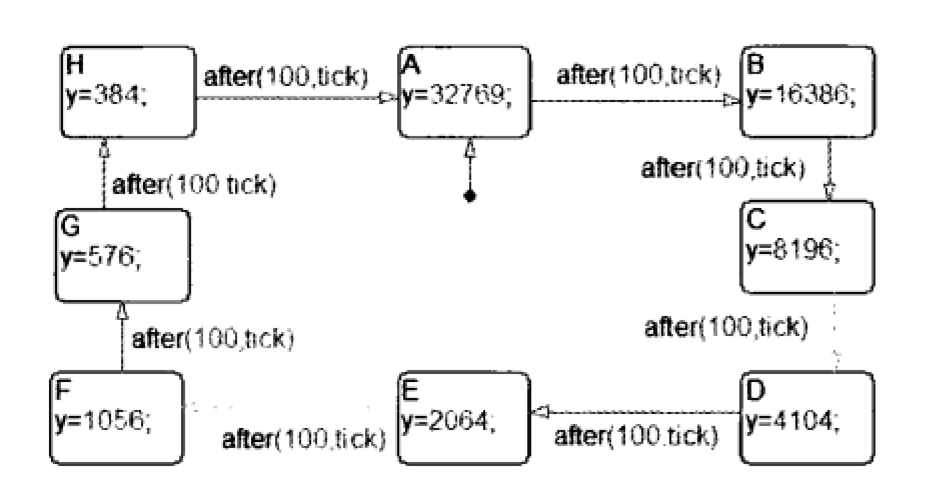


图 7.2.22　闪烁灯驱动模型

2. 闪烁灯功能验证模型

完成闪烁灯驱动模型之后，在 Simulink 模块库中找到图 7.2.23、图 7.2.24 所示的各模块，并按图 7.2.25 所示连接。

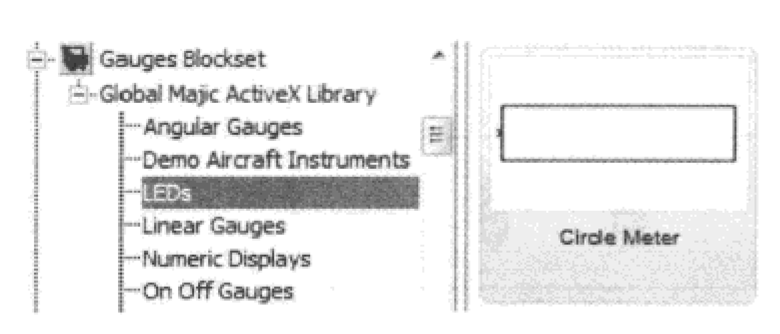


图 7.2.23　Circle Meter 模块

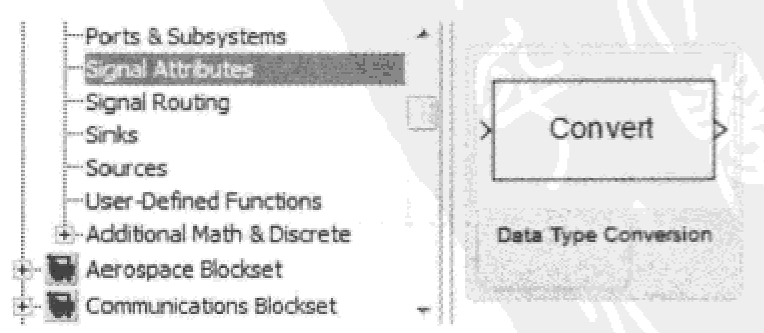


图 7.2.24　数据类型转换模块

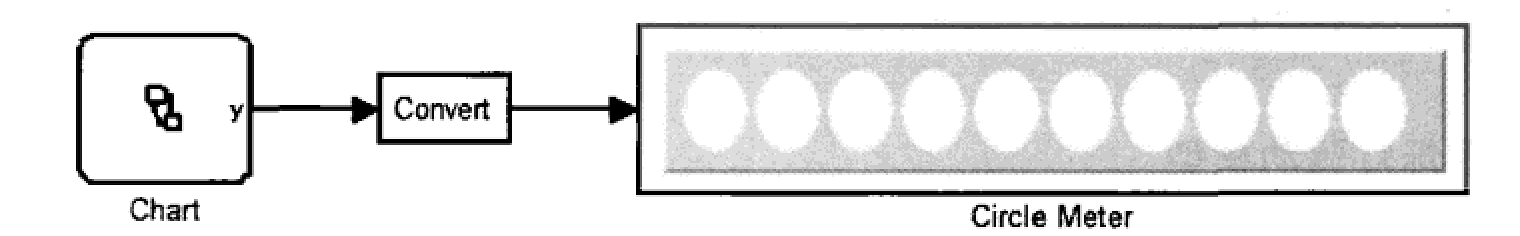


图 7.2.25　功能验证模型

选择模型主窗口的菜单项 Simulation→Configuration Parameters…，打开模型参数对话框，在 Solver 界面中，设置求解器为定步长离散求解器，步长为 0.01，如图 7.2.26 所示。

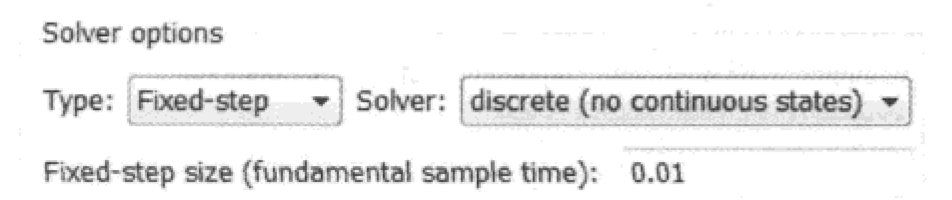


图 7.2.26　求解器设置

设置 Circle Meter 模块 LED 灯数量为 16，输入数据模式为 1-Bitwise，如图 7.2.27 所示。

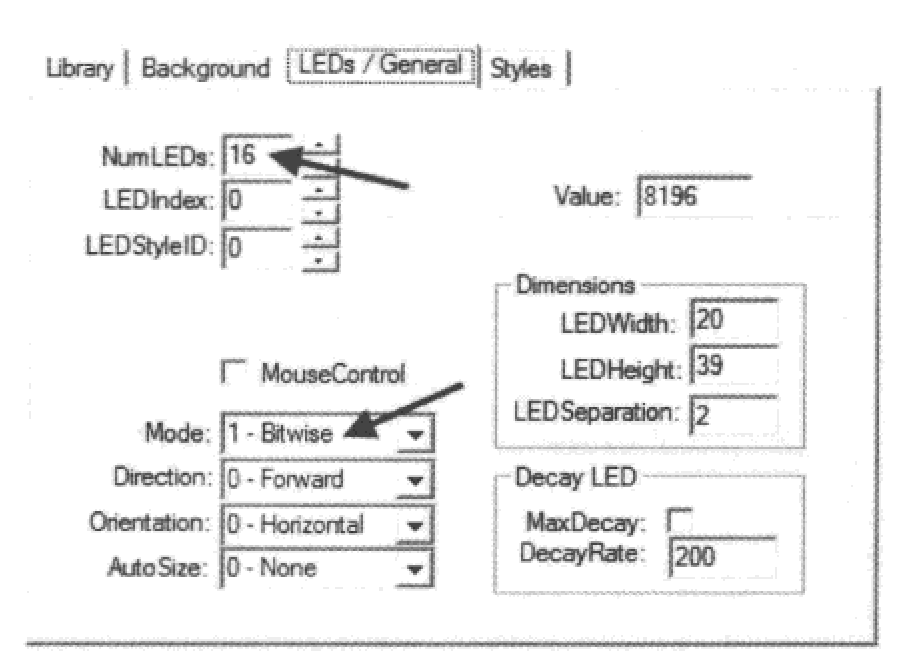


图 7.2.27　Circle Meter 模块设置

完成以上设置后执行仿真，即得到设计所需的亮灯图样，如图 7.2.28、图 7.2.29 所示。

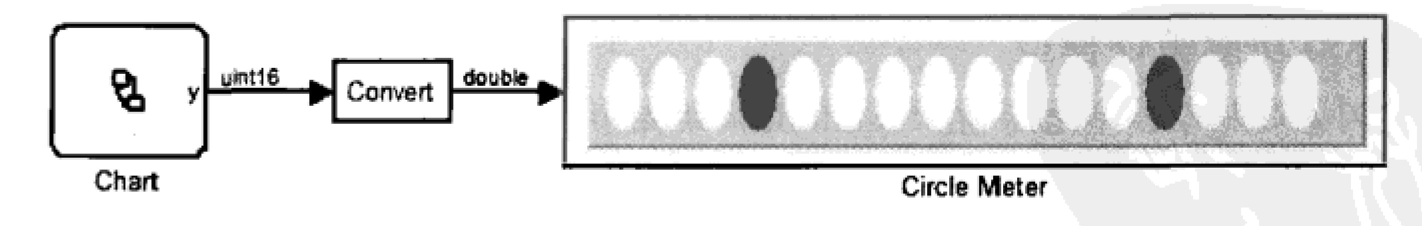


图 7.2.28　功能仿真结果

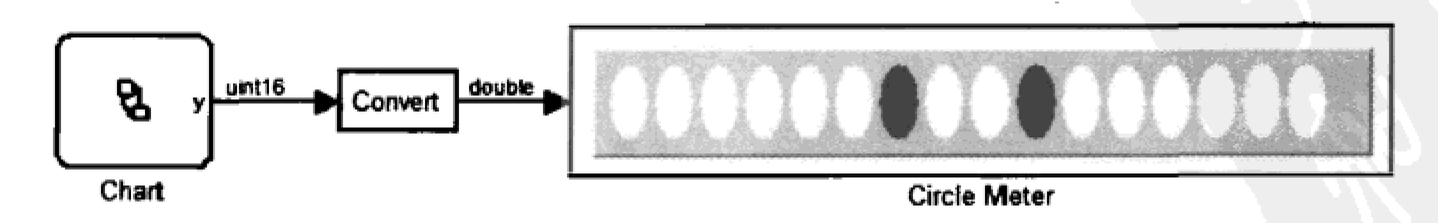
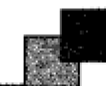


图 7.2.29　功能仿真结果



3. 软件在环测试(SIL)

软件在环测试(SIL)是在主机上对仿真中生成的函数或手写代码进行非实时性联合仿真评估，当软件组件包含需要在目标平台上执行的生成代码和手写代码的组合时，应该考虑进行软件在环测试，完成对模型生成代码的早期验证。

软件在环测试不需要硬件，只是对算法代码进行测试，具体做法是对要进行测试的子系统编译可生成 SIL 模块，比较原模块与 SIL 模块的输出，以此确认算法的正确性。

(1) 数据类型转换。在模块库 Simulink→Ports & Subsystems 中找到模块，替换图 7.2.25 中的 Circle Meter 模块，并将模型另存。

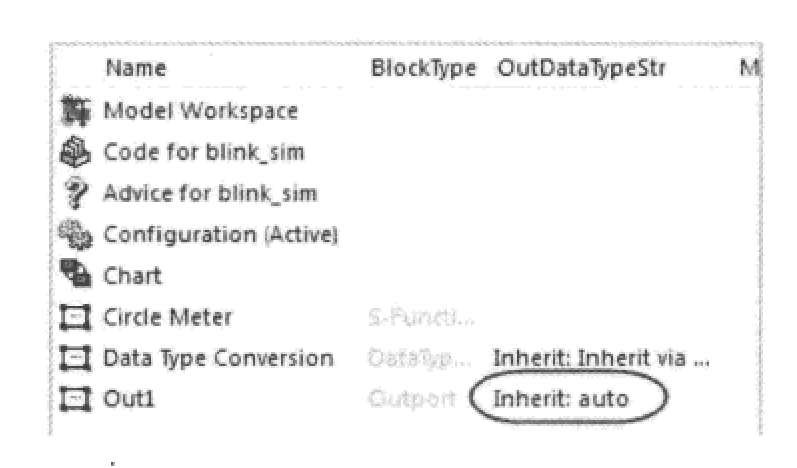


图 7.2.30　修改端口数据类型

单击模型窗口的按钮，打开模型浏览器，闪烁灯驱动模型里变量的数据类型已设置为 uint16，Simulink 模型中的 Out 模块的数据类型可设为自动继承，也可强制设置为 uint16，如图 7.2.30 所示。

修改后的模型如图 7.2.31 所示。

(2) 模型参数设置。打开模型参数对话框，在 Real-Time Workshop 界面设置 TLC 文件为 ert.tlc，如图 7.2.32 所示。

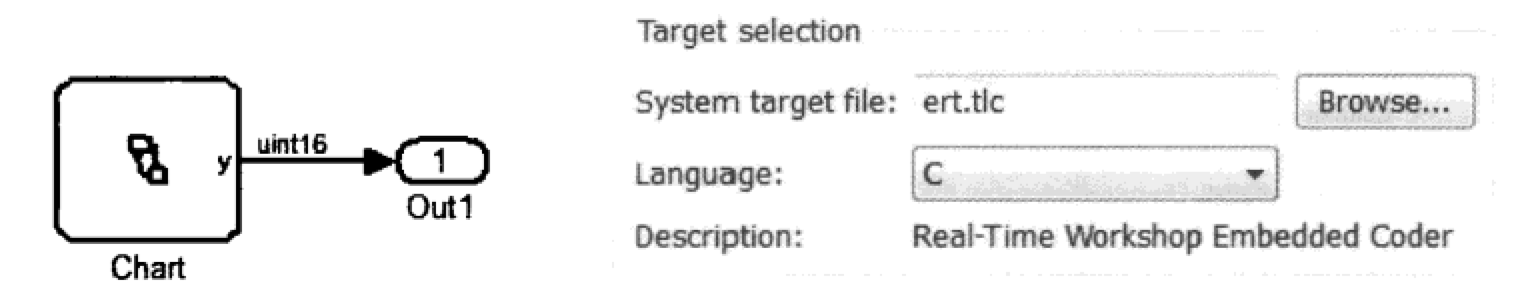


图 7.2.31　代码生成模型

图 7.2.32　设置 TLC

在 Real-Time Workshop→Interface 界面，取消勾选不必要的复选框，如图 7.2.33 所示。

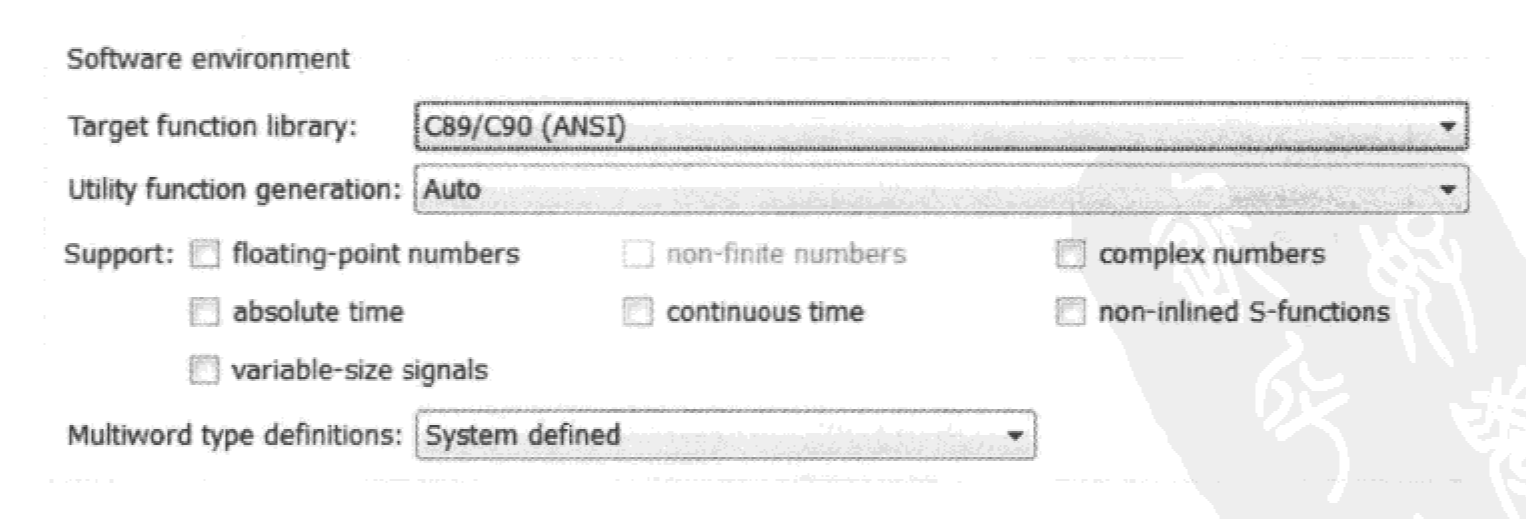
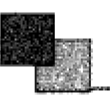


图 7.2.33　Interface 页面设置

在 Real-Time Workshop→Report 界面，勾选所有复选框，便于后期检查及跟踪，如图 7.2.34 所示。



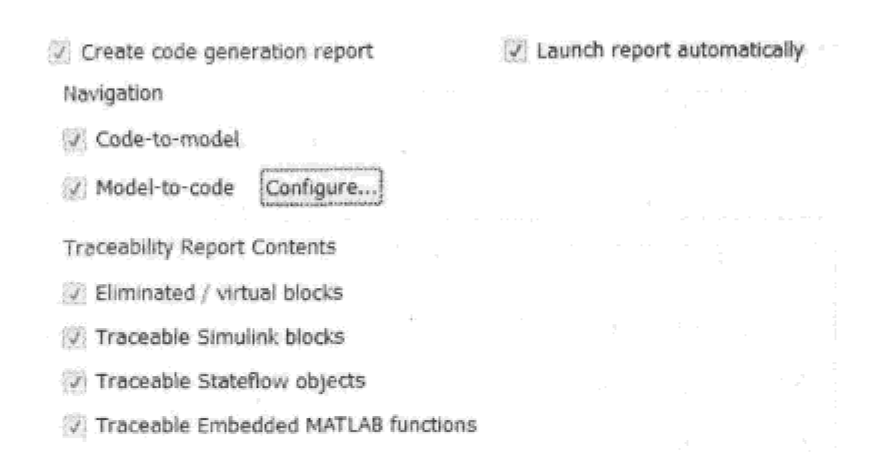


图 7.2.34 报告界面设置

(3) 生成 SIL 模块。在 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表，选择 SIL 选项，如图 7.2.35 所示。

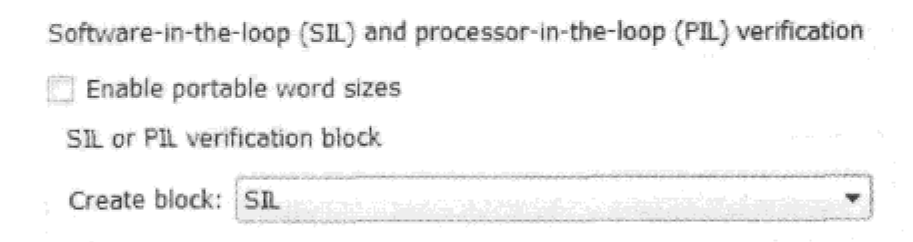


图 7.2.35 SIL 设置

之后单击模型工具栏的按钮，得到代码生成报告如图 7.2.36 所示，SIL 模块如图 7.2.37 所示。

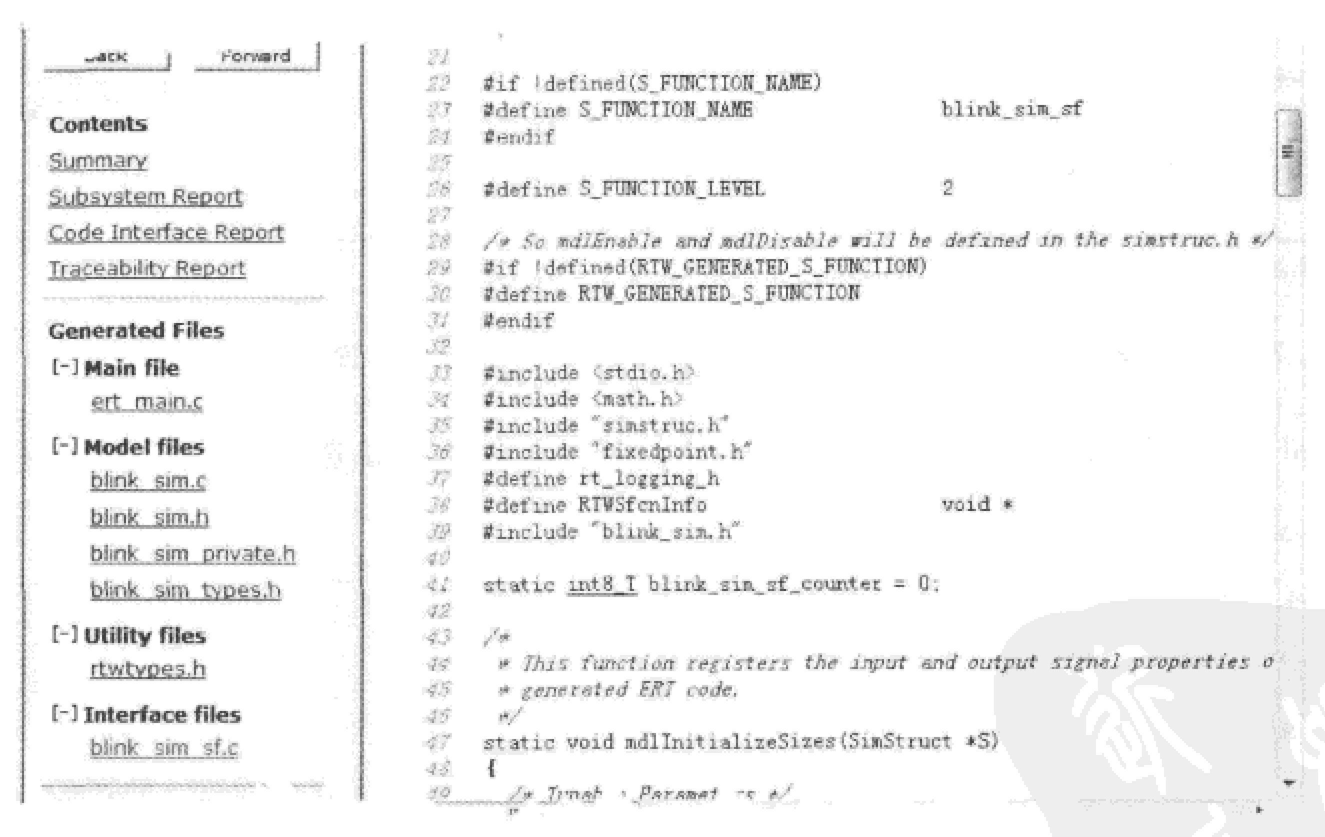


图 7.2.36 代码生成报告

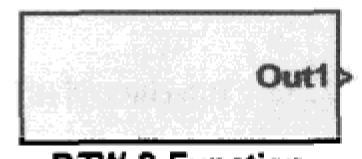
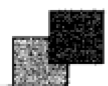


图 7.2.37 SIL 模块



如图 7.2.25 所示，以 SIL 模块替换原有的闪烁灯驱动模型，重建验证模型，该模型的运行结果与功能验证模型是一致的，说明自动生成的代码能实现驱动模型的功能，如图 7.2.38 所示。

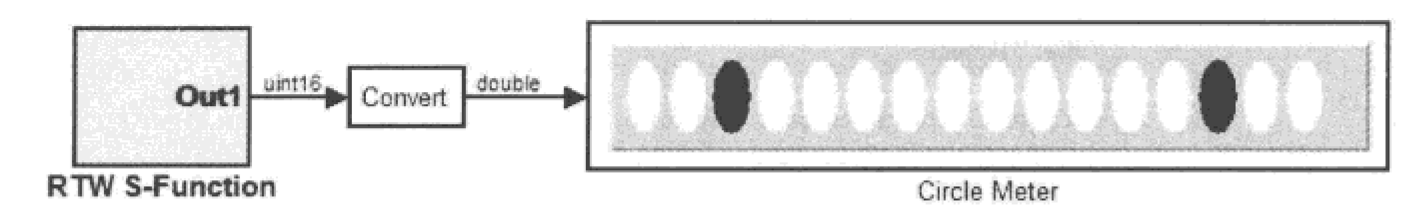


图 7.2.38　软件测试结果

4. 代码生成模型及设置

在 Simulink→ Target for Microchip dsPIC 子模块库找到以下模块，如图 7.2.39～图 7.2.42所示，并按图 7.2.43 所示连接。

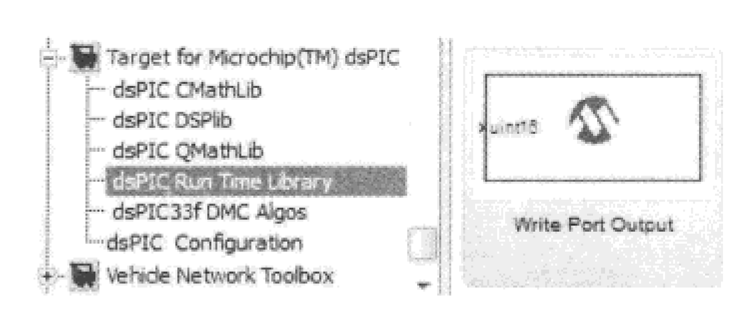


图 7.2.39　Write Port Out put 模块

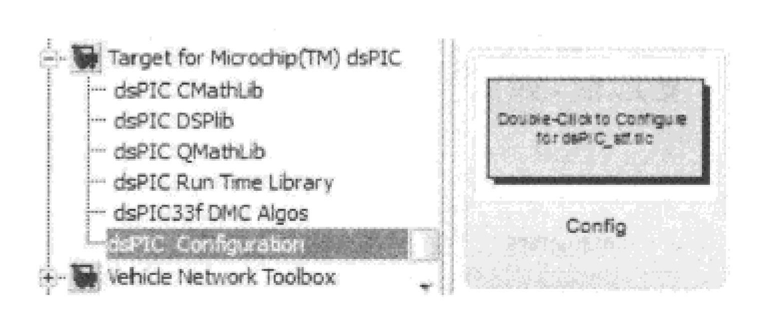


图 7.2.40　Config 模块

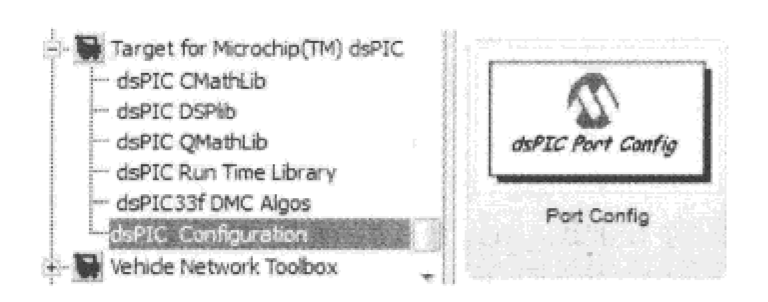


图 7.2.41　Port Config 模块

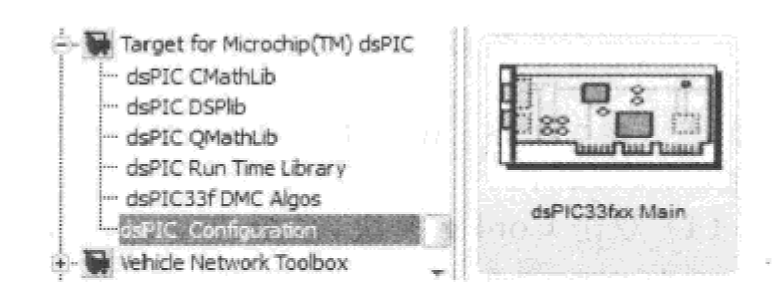


图 7.2.42　dsPIC33fxx Main 模块

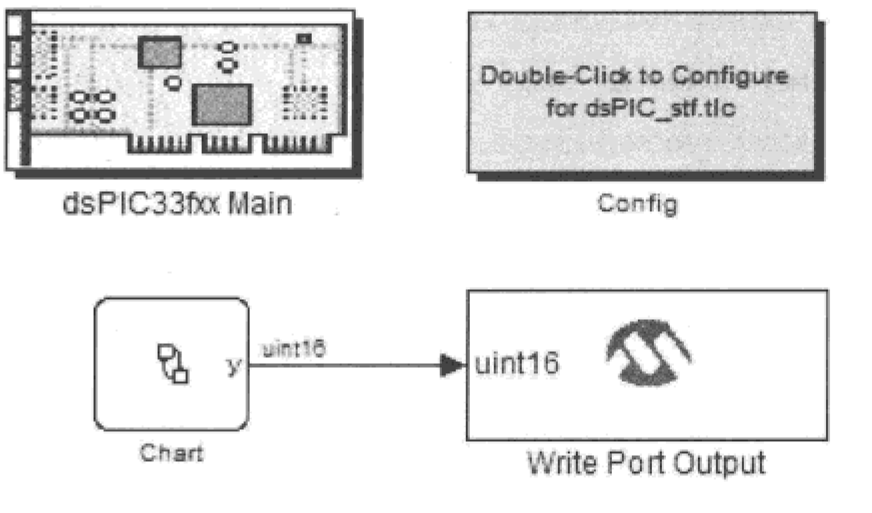


图 7.2.43　闪烁灯模型

(1) 双击 dsPIC33fxx Main 模块，选择振荡源为 Low power RC oscillator，Fcy 显示默认值 16384，如图 7.2.44 所示。

(2) 双击 Write Port Output 模块，选择输出口为 B，取消勾选 Write to selected Pin/s 复选框，如图 7.2.45 所示。

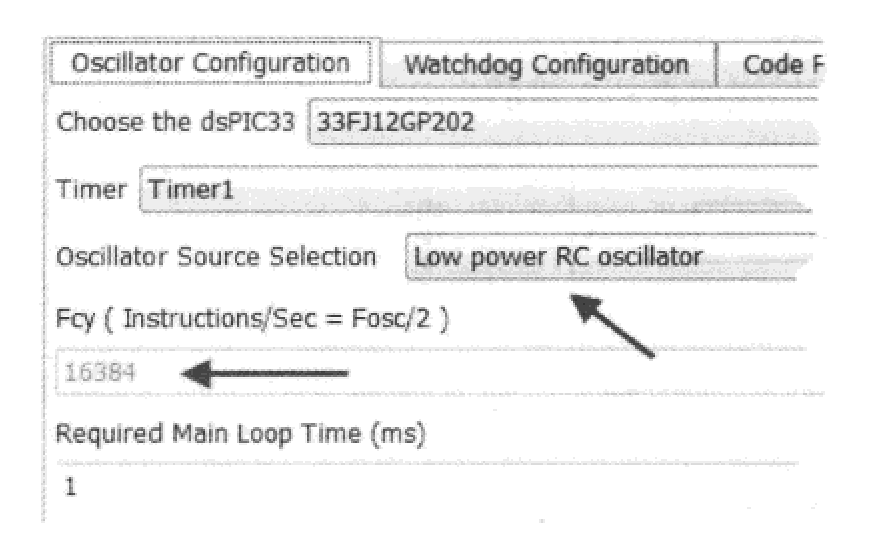


图 7.2.44　dsPIC33fxx Main 模块设置

(3) 双击 Port Config 模块,选择端口 B 的方向为 Output,取消勾选 Configure selected Pin/s 复选框,如图 7.2.46 所示。

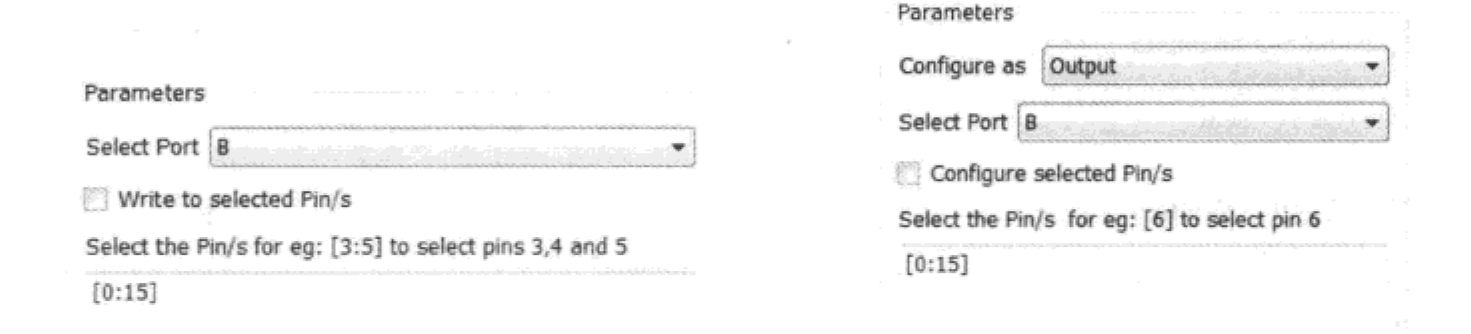


图 7.2.45　Write Port Output 模块设置　　　图 7.2.46　Port Config 模块设置

(4) 双击 Config 模块,系统自动设置 TLC 文件为 dsPIC_stf.tlc,并在模型参数对话框中添加一个 dsPIC Options 界面。在生成代码前,用户应事先检查该界面所有条目对应的文件是否存在,如图 7.2.47 所示。



图 7.2.47　Config 模块设置

(5) 在 Hardware Implementation 界面中,选择硬件设备为 Microchip 公司的 dsPIC,如图 7.2.48 所示。

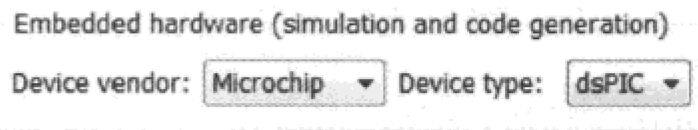
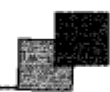


图 7.2.48　选择芯片



(6) 将 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表，选择 none 选项，如图 7.2.49 所示。

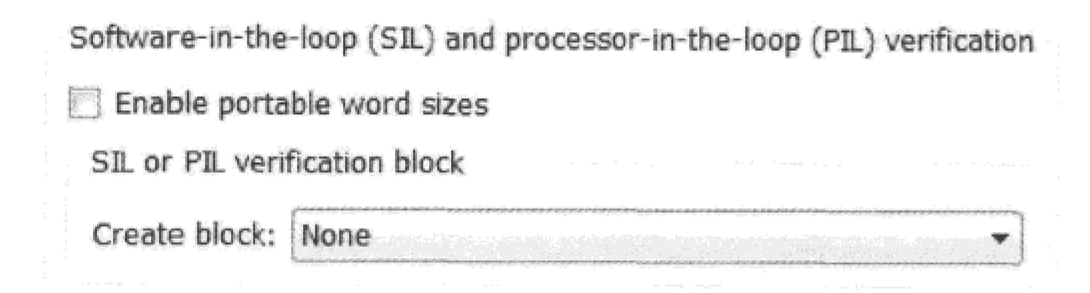


图 7.2.49 SIL/PIL 设置

(7) 单击模型工具栏的按钮，生成代码，报告如图 7.2.50 所示。

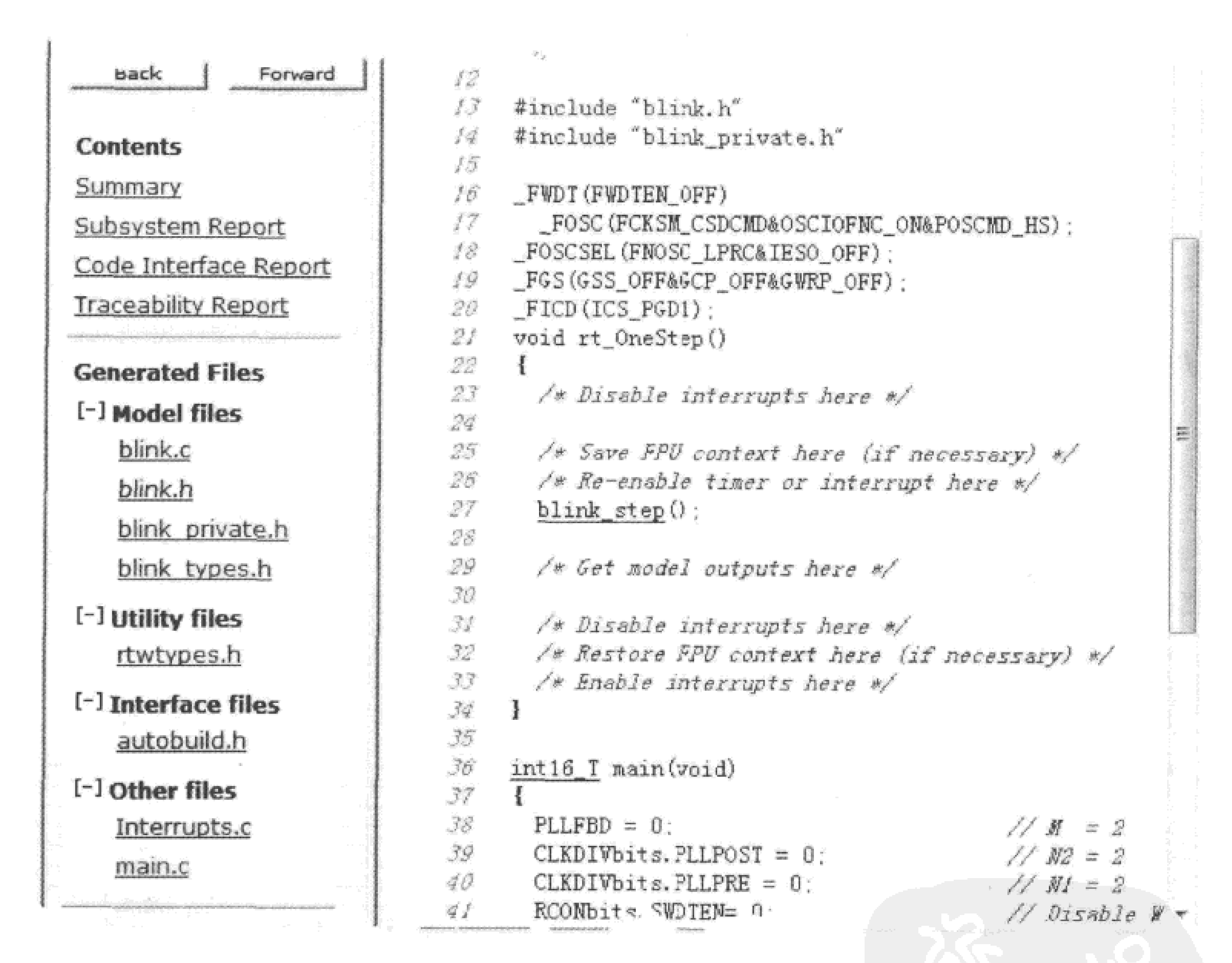


图 7.2.50 代码生成报告

5. 虚拟硬件测试

不同于第 5 章，dsPIC 的自动代码生成过程并不打开 IDE 编译环境，而是直接生成 HEX 文件，生成的源代码保存在模型当前目录的 modelname_dspic_ert 子目录下。

当 MATLAB 命令行显示大致如图 7.2.51 所示时，即生成了 modelname. hex 与 modelname. cof 文件。

```
File  Edit  Debug  Desktop  Window  Help

Data Memory  [Origin = 0x800, Length = 0x400]

section              address     alignment gaps    total length  (dec)
-------              -------     --------------    ------------------
.nbss                  0x800                  0             0x8  (8)
.ndata                 0x808                  0             0x2  (2)

                  Total data memory used (bytes):           0xa  (10) <1%


Dynamic Memory Usage

region               address                      maximum length  (dec)
------               -------                      ---------------------
heap                       0                                 0  (0)
stack                  0x80a                             0x3f6  (1014)

                  Maximum dynamic memory (bytes):         0x3f6  (1014)

"*** Created "executable": ..\../blink.cof"
pic30-bin2hex -omf=coff ..\../blink.cof
### Successful completion of Microchip Embedded Target build procedure for model: blink
fx >>
                                                                    OVR
```

图 7.2.51　编译信息

搭建 proteus 模型，加载生成的 HEX 文件，单击“仿真”按钮，得到与功能验证模型一致的亮灯图样，如图 7.2.52、图 7.2.53 所示。

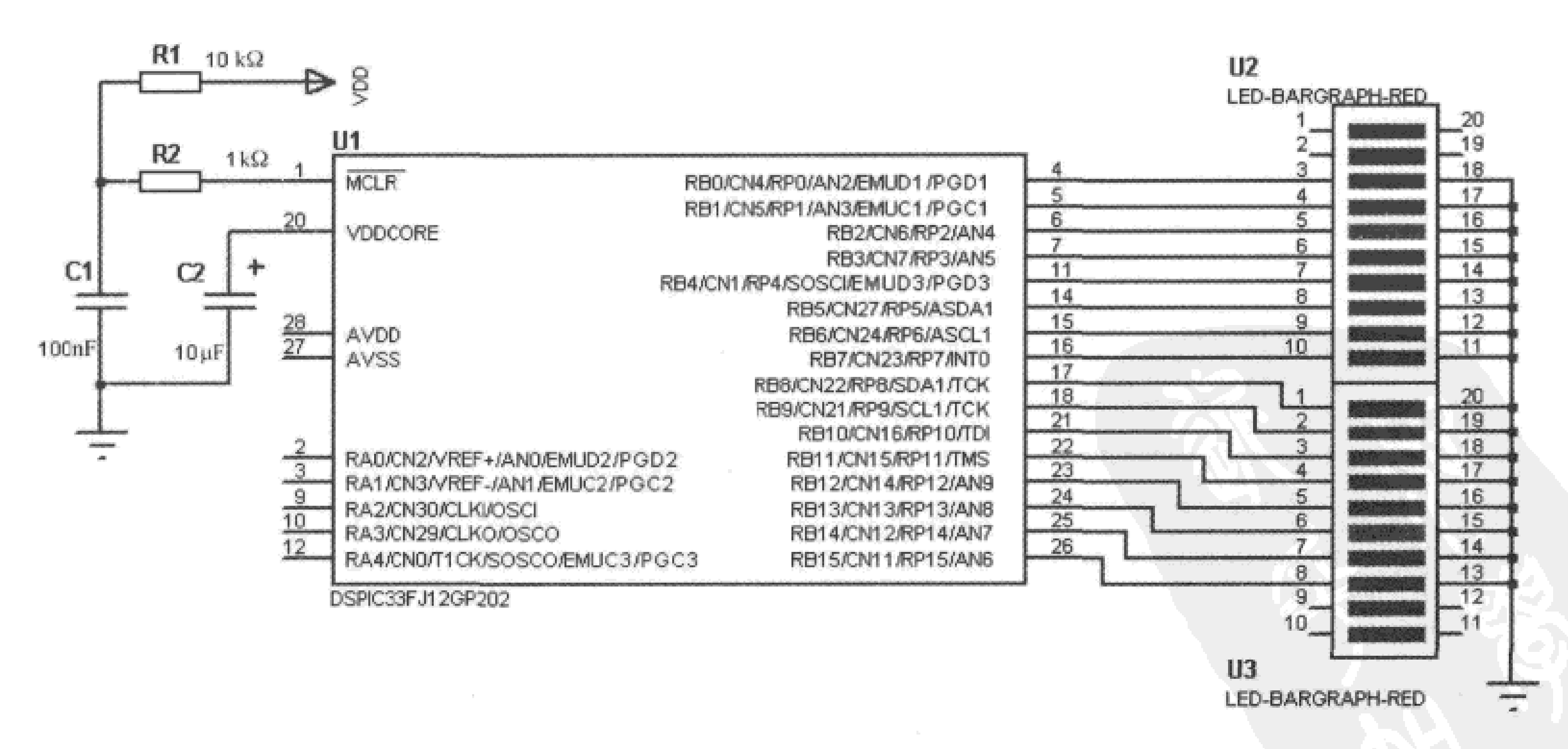
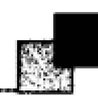


图 7.2.52　仿真结果



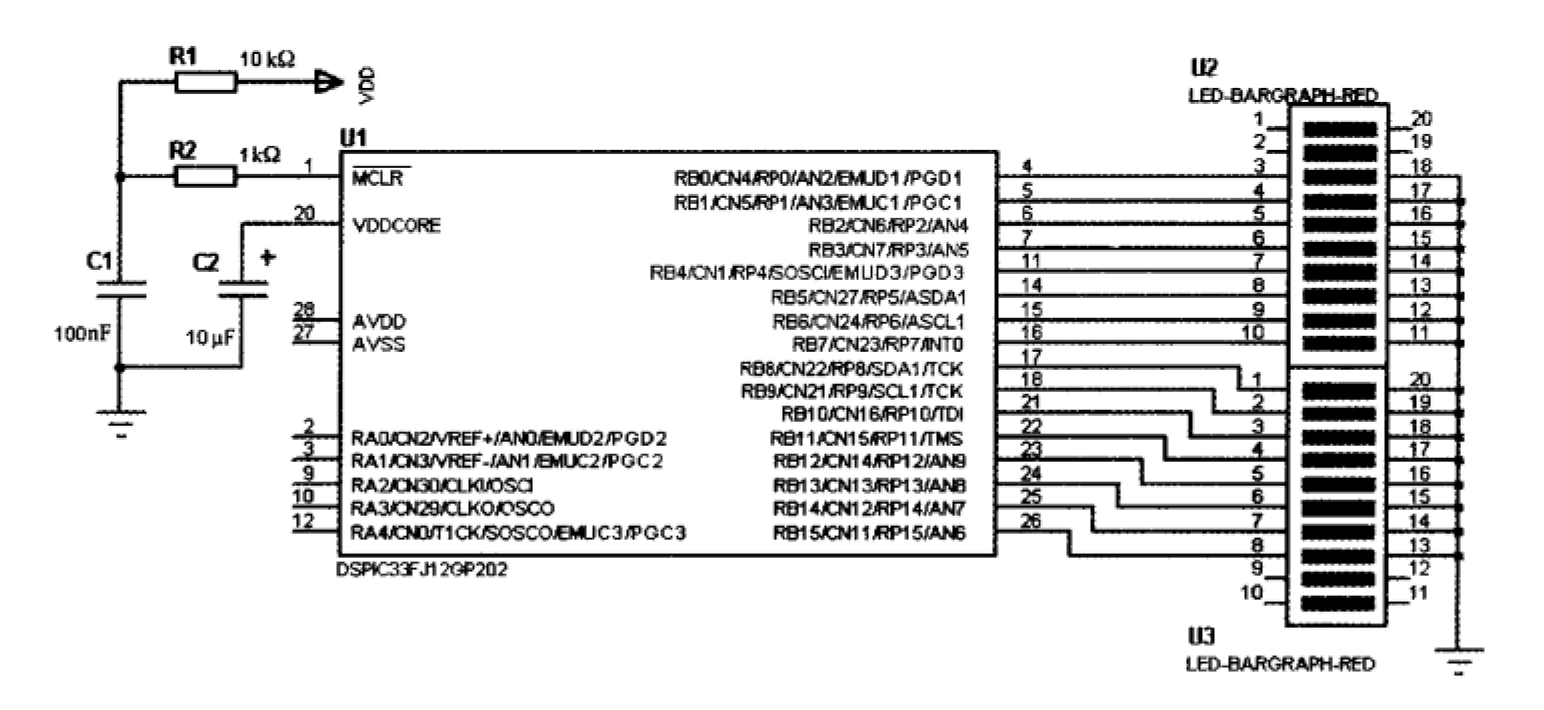
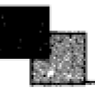


图 7.2.53　仿真结果

用户也可以用逻辑分析仪替换 LED 灯，如图 7.2.54 所示，详细分析亮灯的时序，如图 7.2.55 所示。

图 7.2.54　逻辑分析仪替换 LED 灯



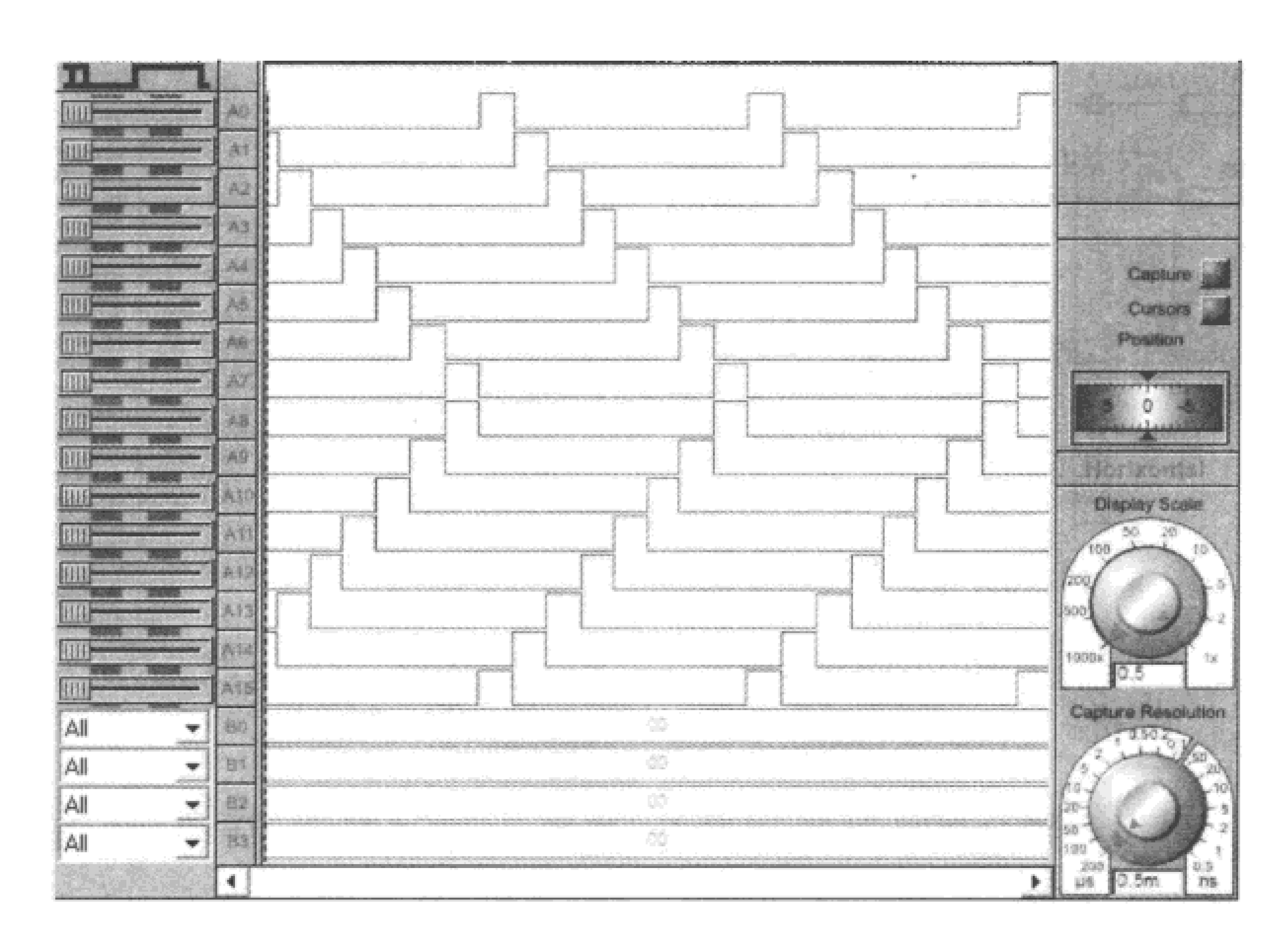


图 7.2.55 逻辑分析仪波形

7.2.3 调用现有 C 函数

1. 创建功能验证模型

在 Simulink 模块库中找到图 7.2.56、图 7.2.57 所示模块，并按图 7.2.58 所示连接，建立一个简单的加法模型，对于实际应用，加法模块可以替换成各种具体的算法。

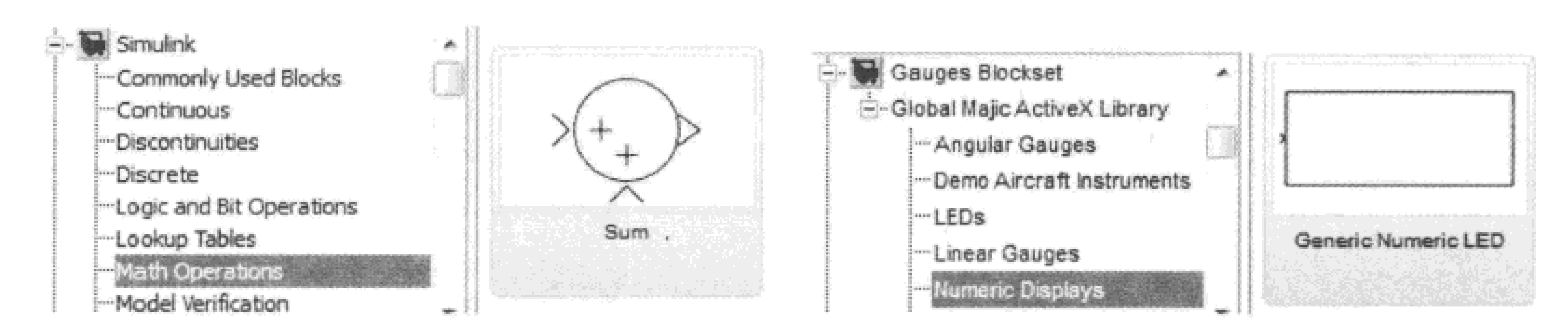


图 7.2.56 Sum 模块 图 7.2.57 Generic Numeric LED 模块

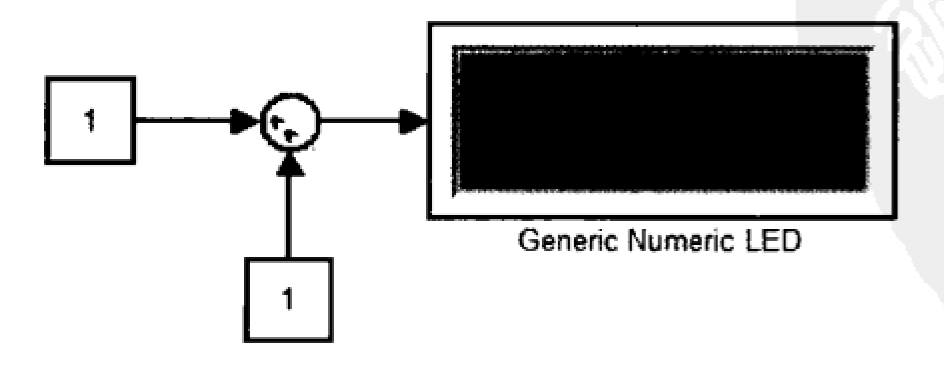
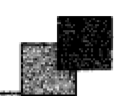


图 7.2.58 功能验证模型



1位十进制整数加法的和最多为2位十进制整数，因此设置数码管的显示位数为2位整数、0位小数，如图7.2.59所示。

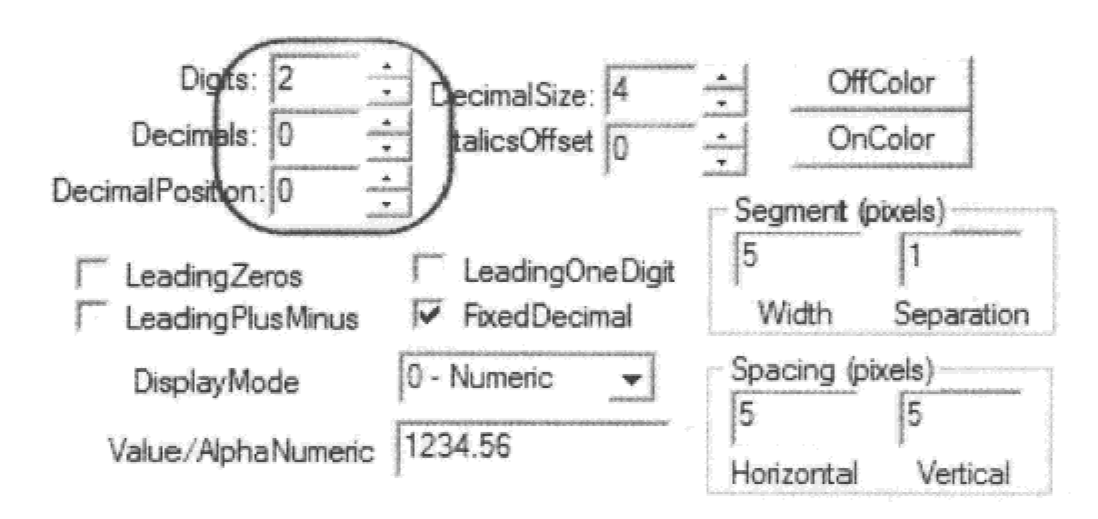


图7.2.59 Generic Numeric LED设置

选择模型主窗口的菜单项Simulation→Configuration Parameters…，打开模型参数对话框，在Solver Options界面中，设置求解器为定步长离散求解器，步长为0.01，如图7.2.60所示。

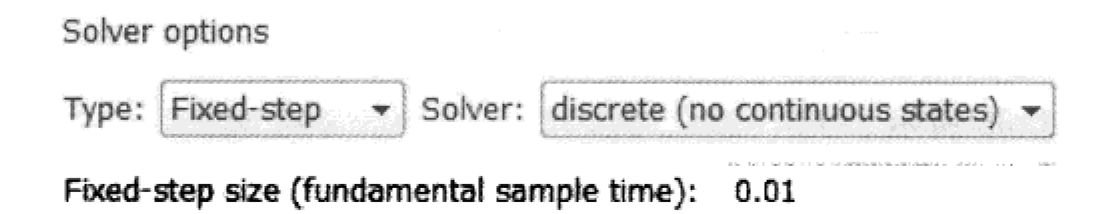


图7.2.60 求解器设置

仿真结果如图7.2.61、图7.2.62所示。

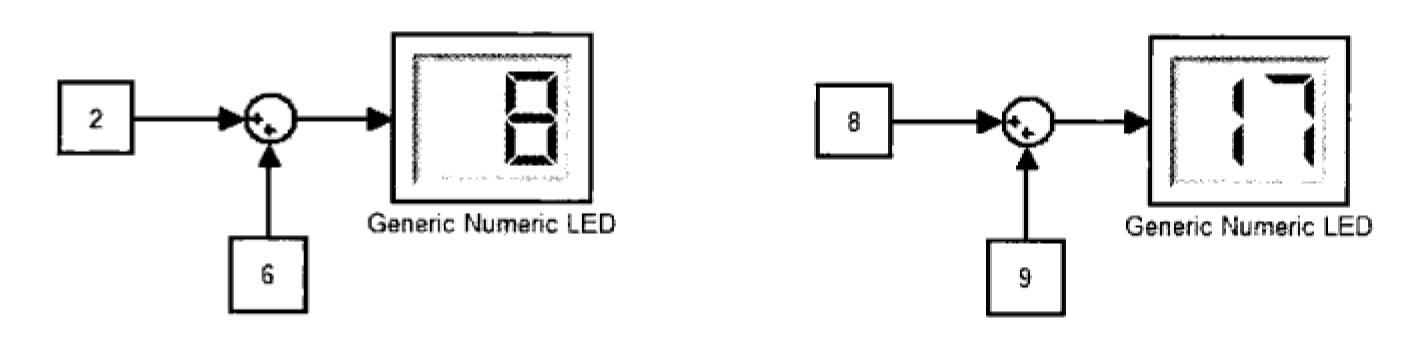


图7.2.61 功能仿真结果(一)　　图7.2.62 功能仿真结果(二)

2. 软件在环测试

(1) 数据类型转换。在模块库Simulink→Ports & Subsystems中找到输入模块与输出模块，替换图7.2.58中的常数与LED模块，并将模型另存。

单击模型窗口的按钮，打开模型浏览器，将功能验证模型中的In模块的数据类型设置为uint16，加法模块的数据类型设置为uint16，Out模块的数据类型可设为自动继承，也可强制设置为uint16，如图7.2.63所示。

修改后的模型如图7.2.64所示。

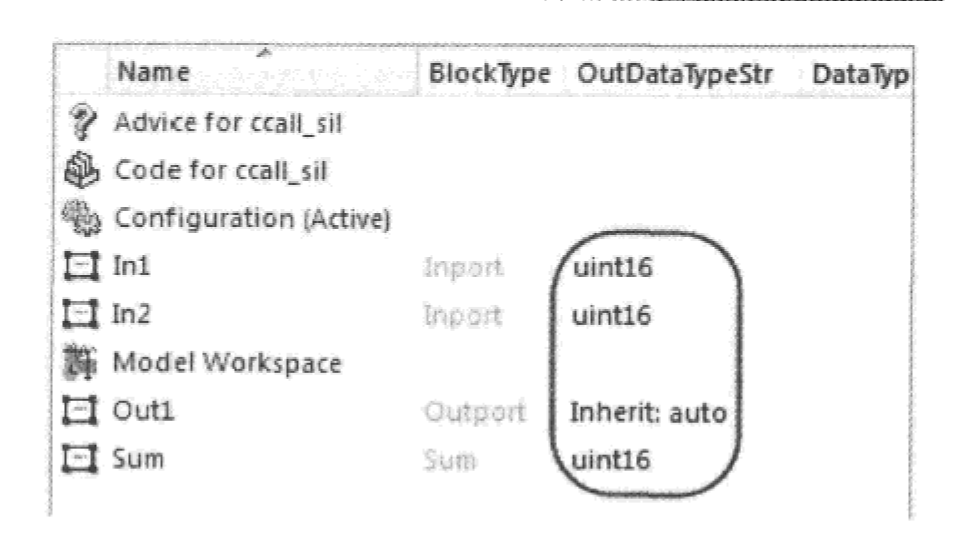


图 7.2.63　修改模型端口数据类型

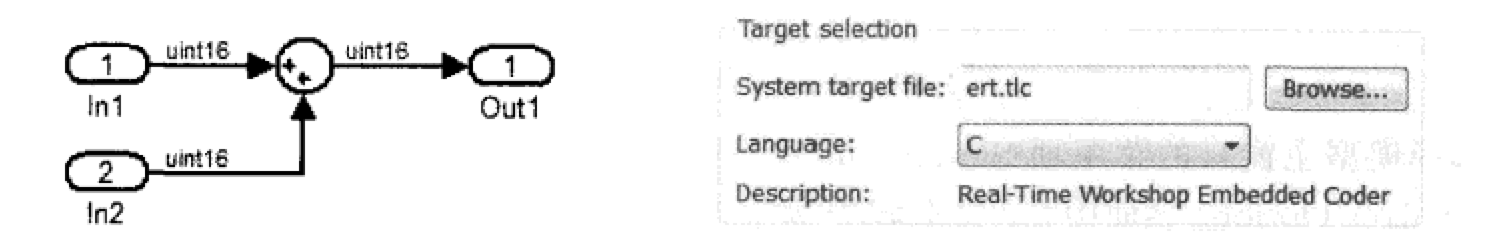


图 7.2.64　代码生成模型　　　　图 7.2.64　TLC 设置

(2) 模型参数设置。打开模型参数对话框，在 Real-Time Workshop 界面设置 TLC 文件为 ert.tlc，如图 7.2.65 所示。

Real-Time Workshop→Interface 界面，取消勾选不必要的复选框，如图 7.2.65 所示。

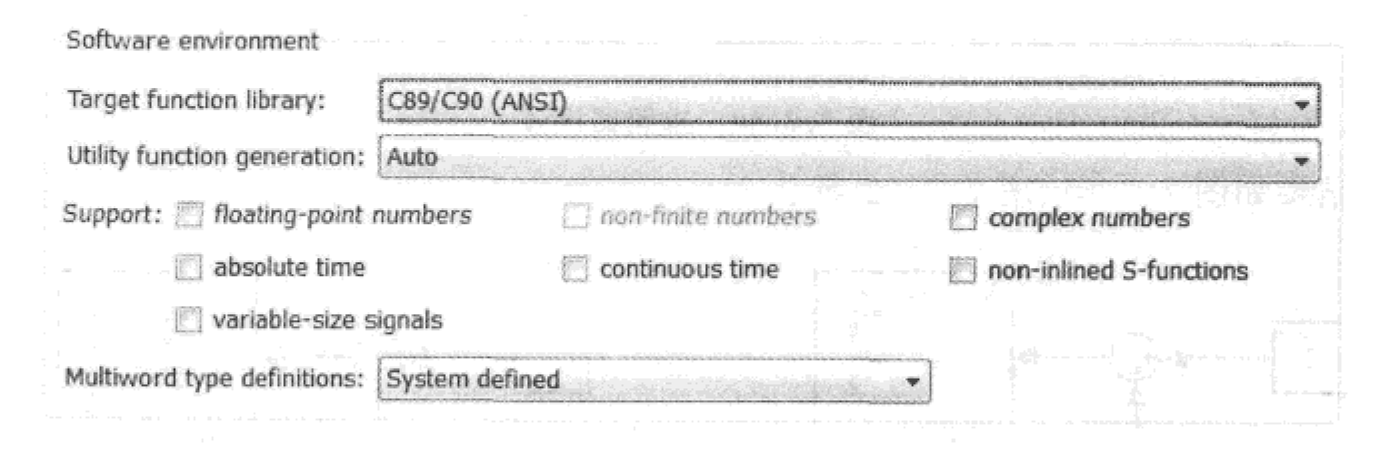


图 7.2.65　Interface 界面设置

Real-Time Workshop→Report 界面，勾选所有复选框，便于后期检查及跟踪，如图 7.2.66 所示。

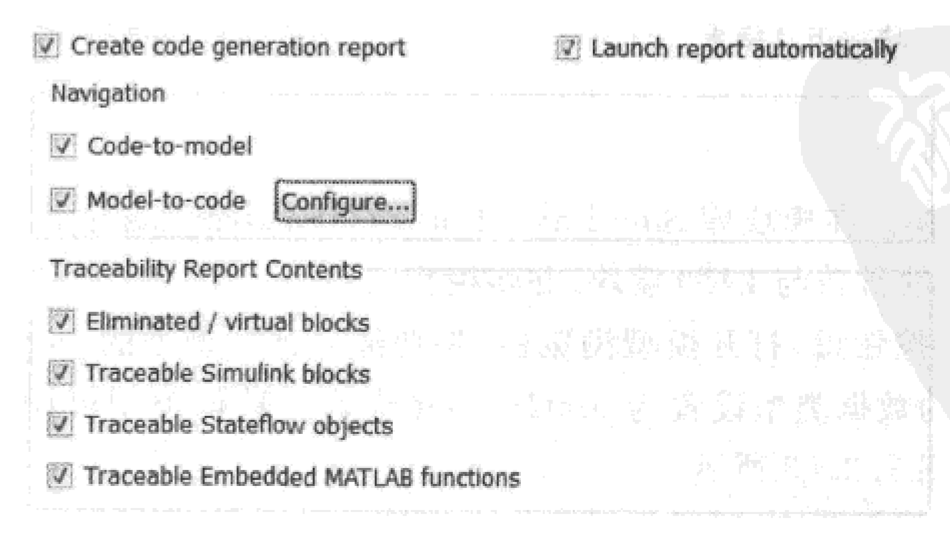


图 7.2.66　报告界面设置



(3) 生成 SIL 模块。在 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表,选择 SIL 选项,如图 7.2.67 所示。

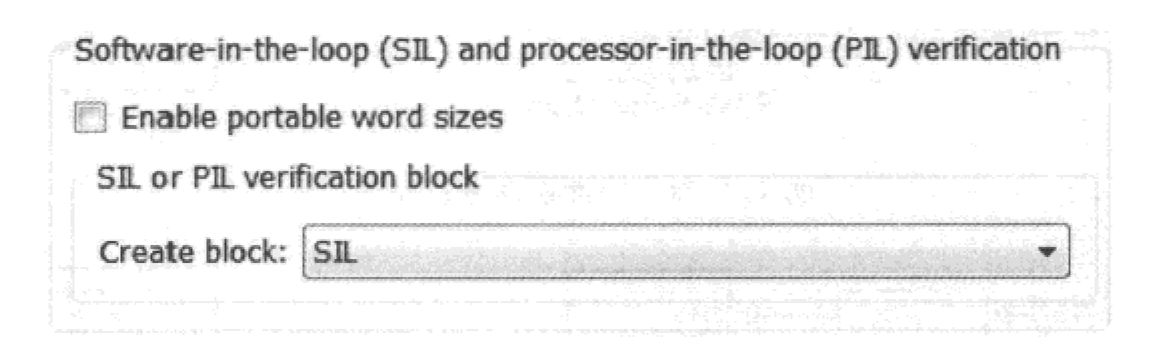


图 7.2.67　SIL 设置

之后单击模型工具栏的按钮,得到代码生成报告如图 7.2.68 所示,SIL 模块如图 7.2.69 所示。

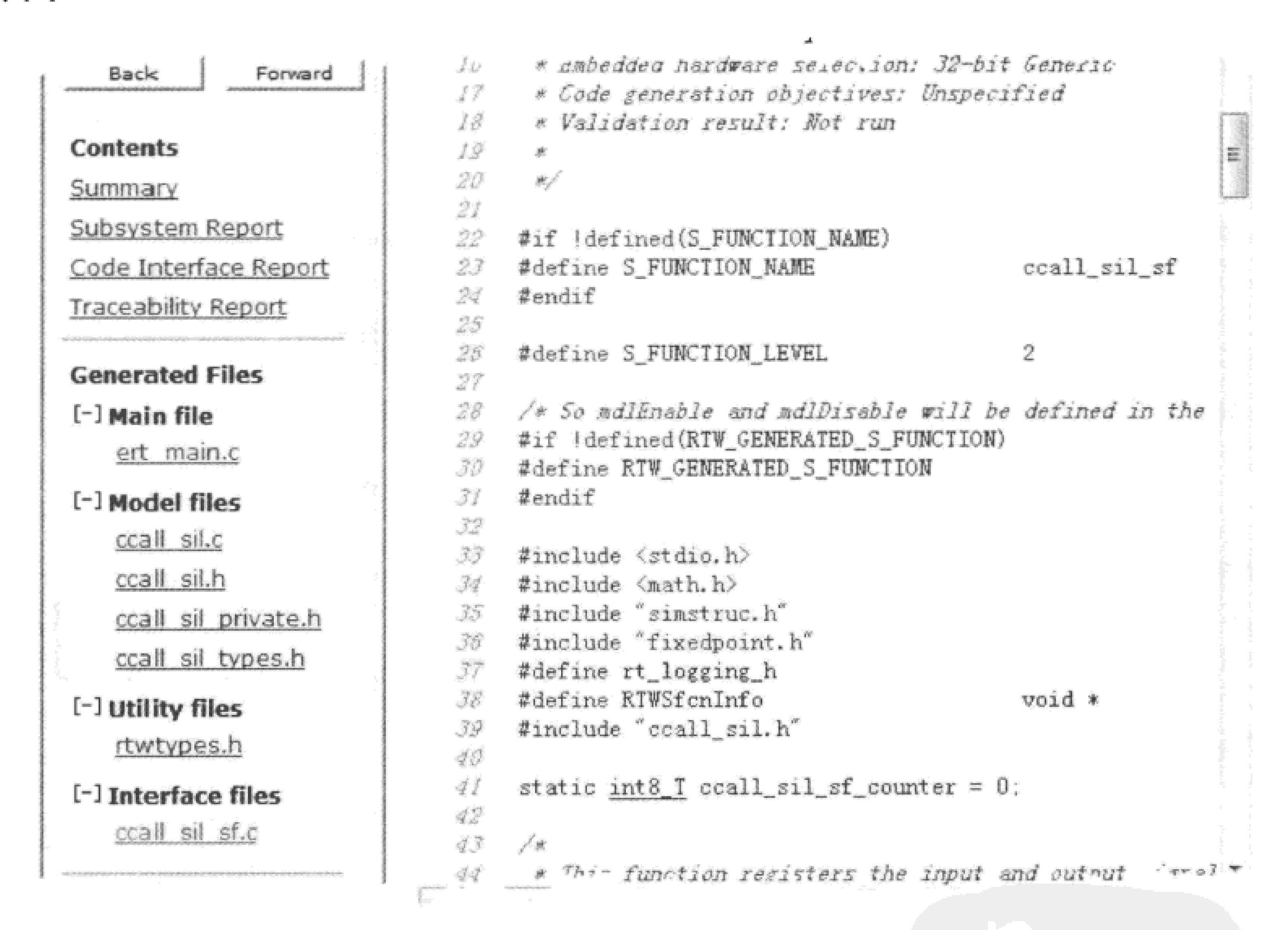


图 7.2.68　代码生成报告

如图 7.2.70、图 7.2.71 所示,以 SIL 模块替换原有的 Product 模块,重建验证模型,并在各端口间加入必要的数据类型转换模块。SIL 测试的结果,如图 7.2.70、图 7.2.71 所示结果与图 7.2.61、图 7.2.62 所示的结果是一致的。说明自动生成的代码可以实现模型的功能。

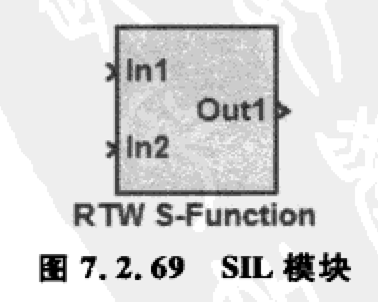


图 7.2.69　SIL 模块

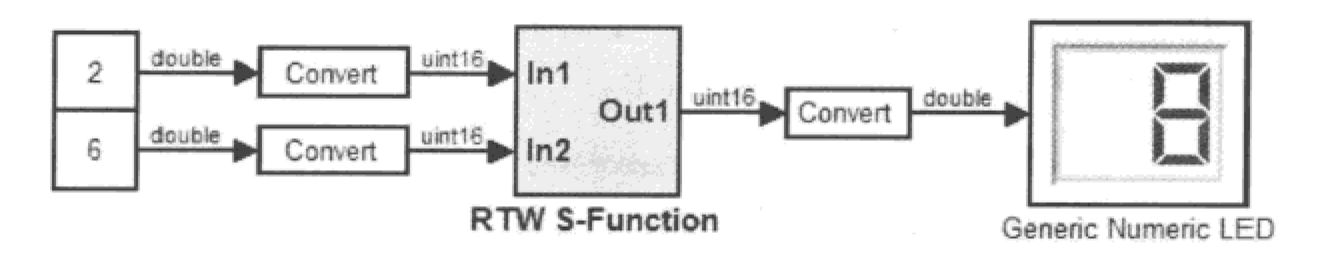


图 7.2.70　软件在环测试结果

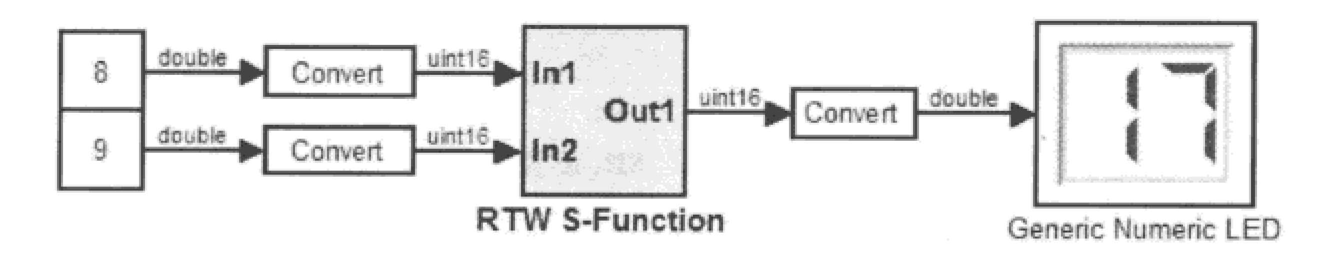


图 7.2.71　软件在环测试结果

3. 代码生成模型及设置

在 Simulink→Target for Microchip dsPIC 子模块库找到图 7.2.72～图 7.2.77 所示的模块，并按图 7.2.78 所示连接。

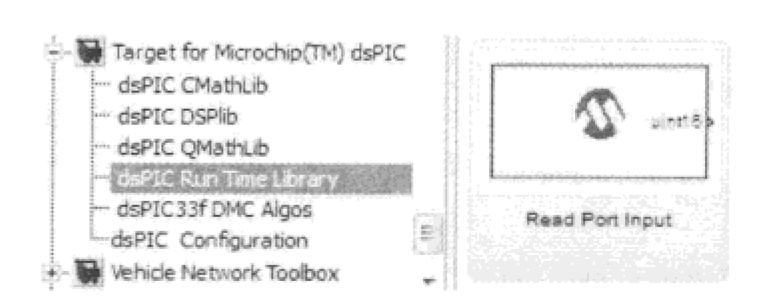


图 7.2.72　Read Port Input 模块

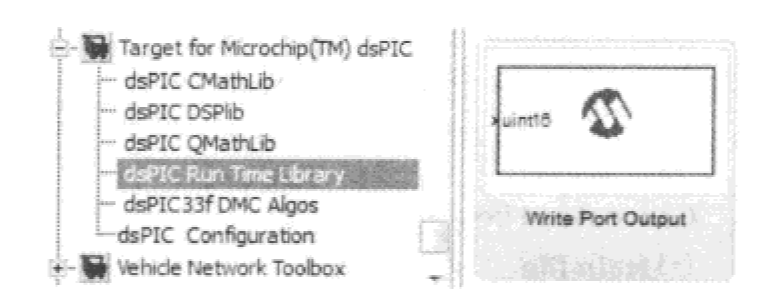


图 7.2.73　Write Port Output 模块

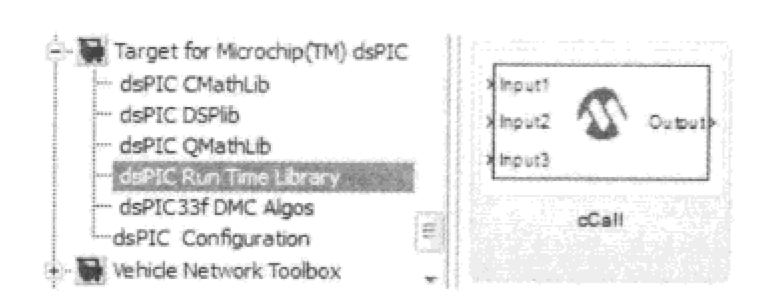


图 7.2.74　cCall 模块

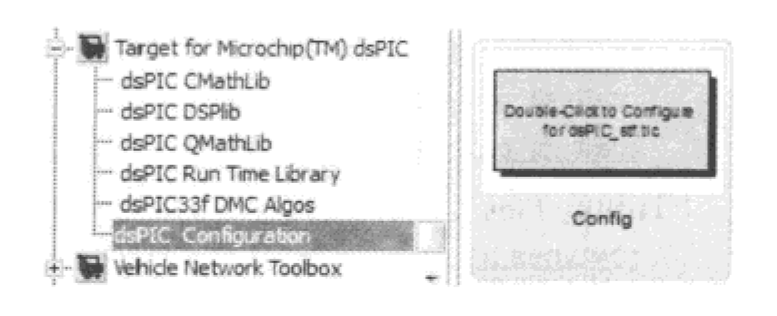


图 7.2.75　Config 模块

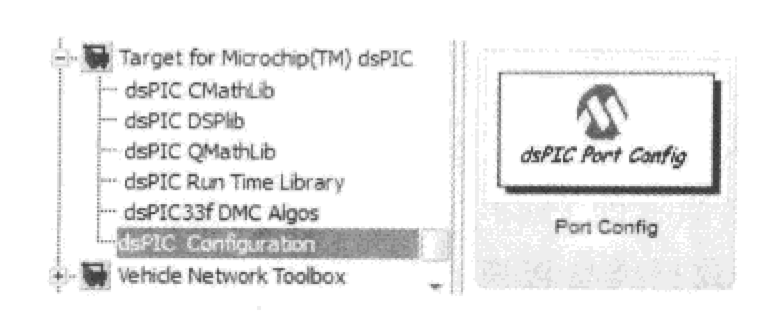


图 7.2.76　Port Config 模块

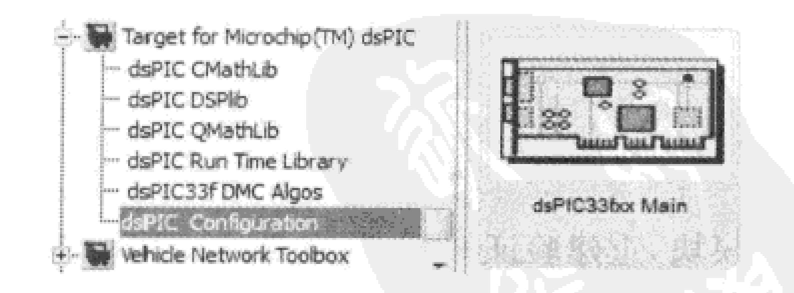
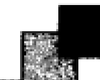


图 7.2.77　dsPIC33fxx Main 模块

单片机的输出端口通常是不能直接显示 2 位以上十进制数的，为此需要将十进制数的每一位分离出来，单独送数码管显示。

以下 C 代码实现了 2 位十进制数的分离，以函数名作为文件名保存，供 cCall 模块调用。



```
int dec(unsigned int x)
{
    unsigned int sum;
    unsigned int a1,a2;
    a1 = x % 10; //计算个位
    a2 = x/10; //计算十位
    sum = a2<<4|a1;
    return sum;
}
```

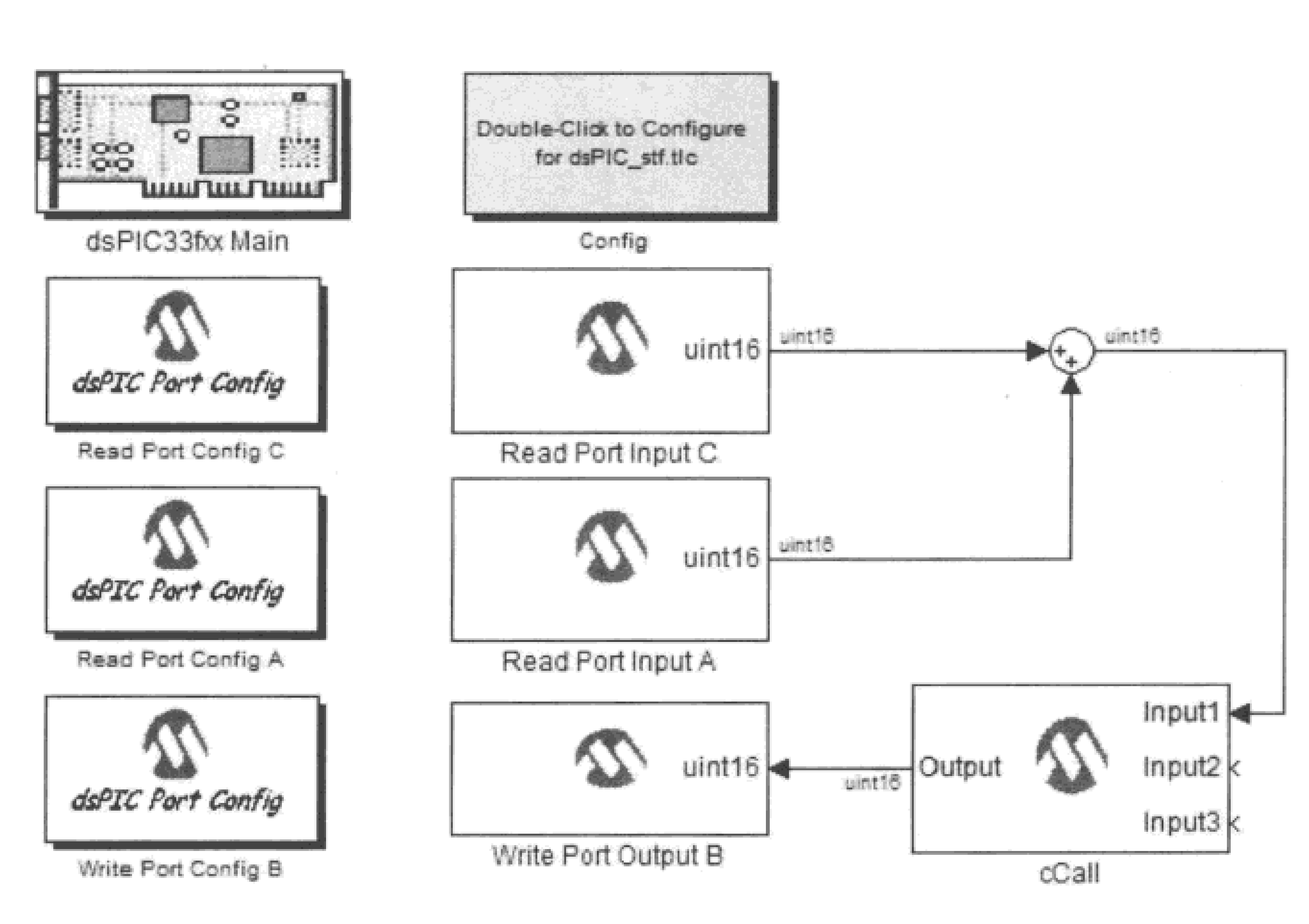


图 7.2.78　Simulink 功能验证模型

模块与模型参数设置与 7.3.1 节无太大差异，这里仅作简单罗列。

(1) dsPIC33fxx Main 模块，选择处理器芯片为 33FJ16GP304，振荡源为 Low power RC oscillator，Fcy 显示默认值 16384，如图 7.2.79 所示。

(2) 双击 cCall 模块，指定需要调用的 C 函数，函数名应使用半角单引号包围，如图7.2.80所示。

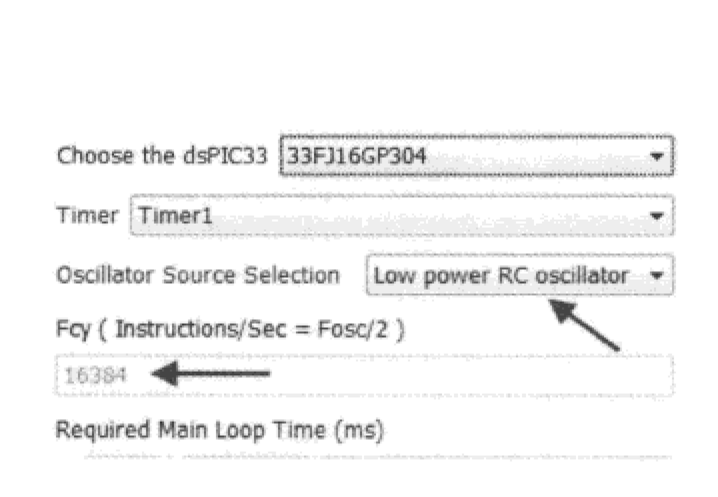


图 7.2.79　dsPIC33fxx Main 模块设置

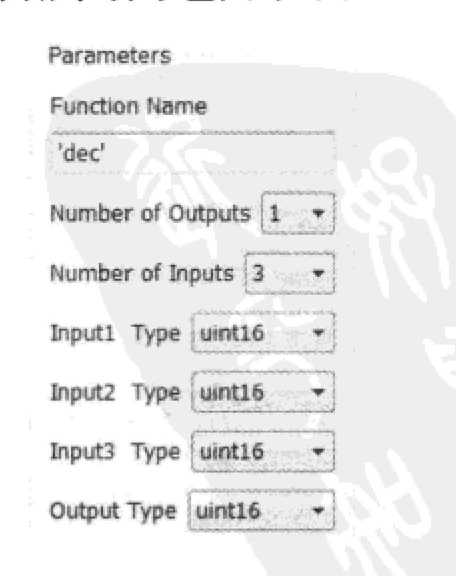


图 7.2.80　cCall 模块设置



(3) Read Port Input。2 个 Read Port Input 模块的输入引脚分别为端口 C、端口 A，不应勾选 Read from selected Pin/s 复选框，如图 7.2.81 所示。

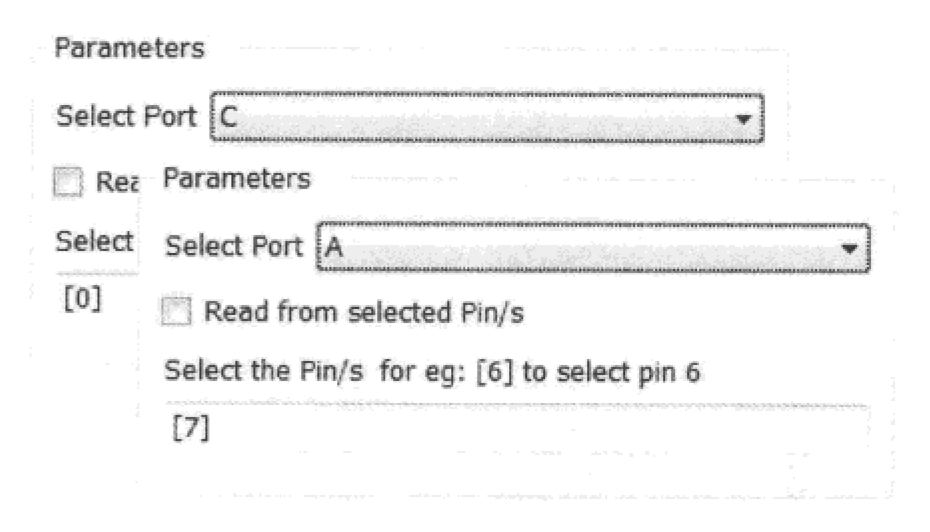


图 7.2.81 Read Port Input 模块设置

(4) Read Port Config。选择端口 C 的方向为输入，勾选 Configure selected Pin/s 复选框，引脚定义为[0:3]，如图 7.2.82 所示。

选择端口 A 的方向为输入，勾选 Configure selected Pin/s 复选框，引脚定义为[7:10]，如图 7.2.83 所示。



图 7.2.82 Read Port Config 模块设置 **图 7.2.83 Read Port Input 模块设置**

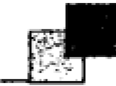
(5) Write Port Output。选择输出引脚为端口 B，取消勾选 Write to selected Pin/s 复选框，如图 7.2.84 所示。

(6) Write Port Config。选择端口 B 的方向为输出，取消勾选 Configure selected Pin/s 复选框，如图 7.2.85 所示。



图 7.2.84 Write Port Output 模块设置 **图 7.2.85 Write Port Config 模块设置**

(7) 双击 Config 模块，系统自动设置 TLC 文件为 dsPIC_stf.tlc，并在模型参数对话框中添加一个 dsPIC Options 界面。在生成代码前，用户应事先检查该界面所有条目对应的文件是否存在，如图 7.2.86 所示。



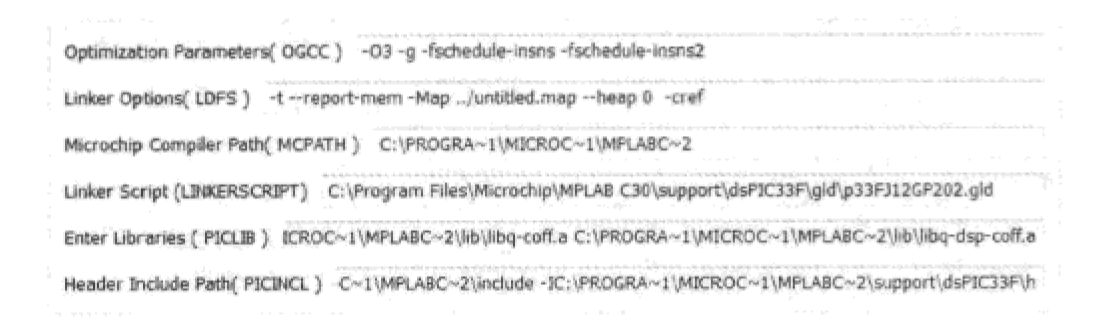


图 7.2.86 dsPIC Options 界面设置

(8) 在 Hardware Implementation 界面，选择硬件设备为 Microchip 公司的 dsPIC，如图 7.2.87 所示。

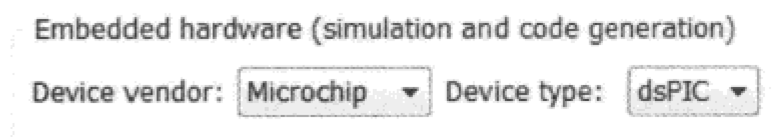


图 7.2.87 选择芯片

(9) 将 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表，选择 none 选项，如图 7.2.88 所示。

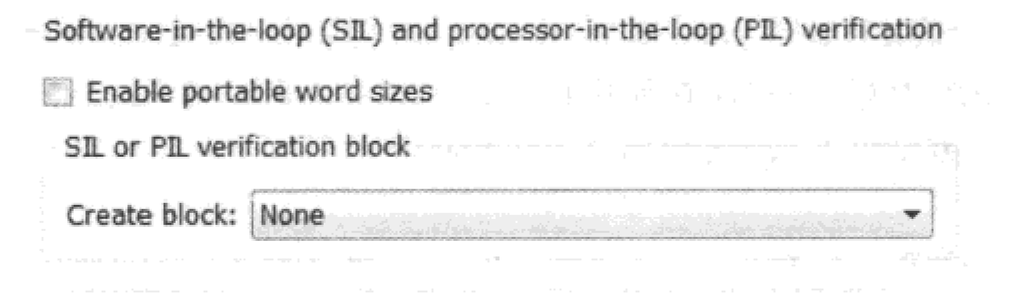


图 7.2.88 Interface 界面设置

4. MPLAB 环境下的代码生成及修改

这时用户可以单击模型工具栏的按钮，生成代码。不过本节使用另一种方法实现这一过程。用户若完全安装了 MPLAB IDE v8.56，软件菜单项 Tools 下应有 Matlab/Simulink 命令，如图 7.2.89 所示。

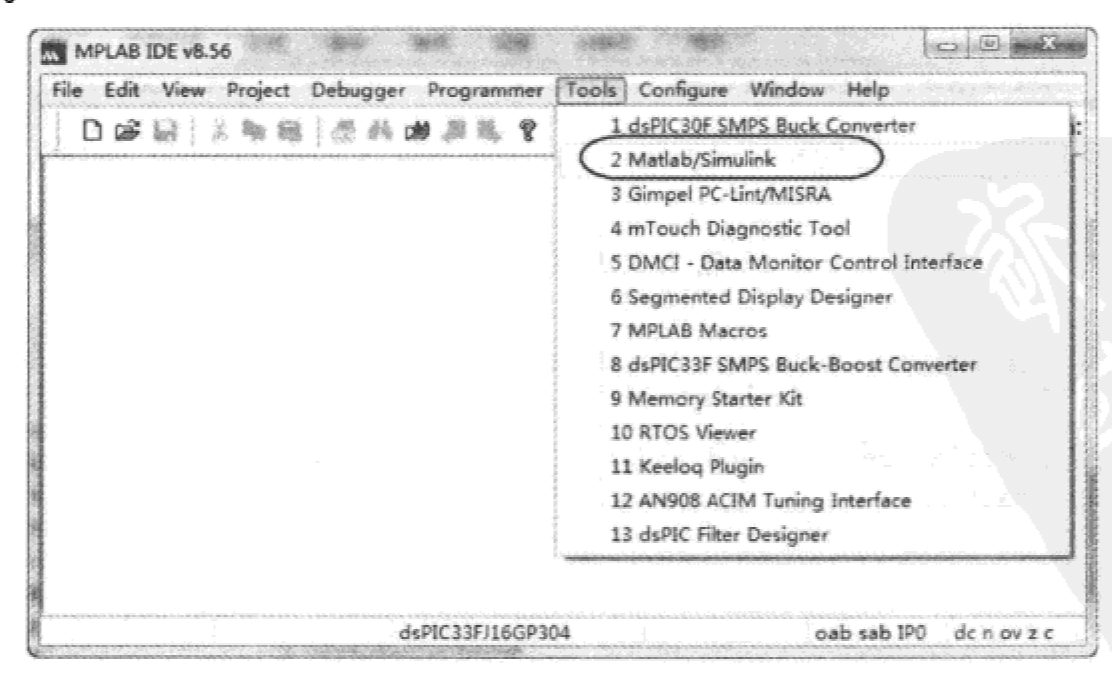


图 7.2.89 Matlab/Simulink 插件

选择此命令之后，菜单栏增加了 Matlab/Simulink 这一项目，选择菜单项 Matlab/Simulink→Specify Simulink Model Name，如图 7.2.90 所示，指定需要生成代码的模型。

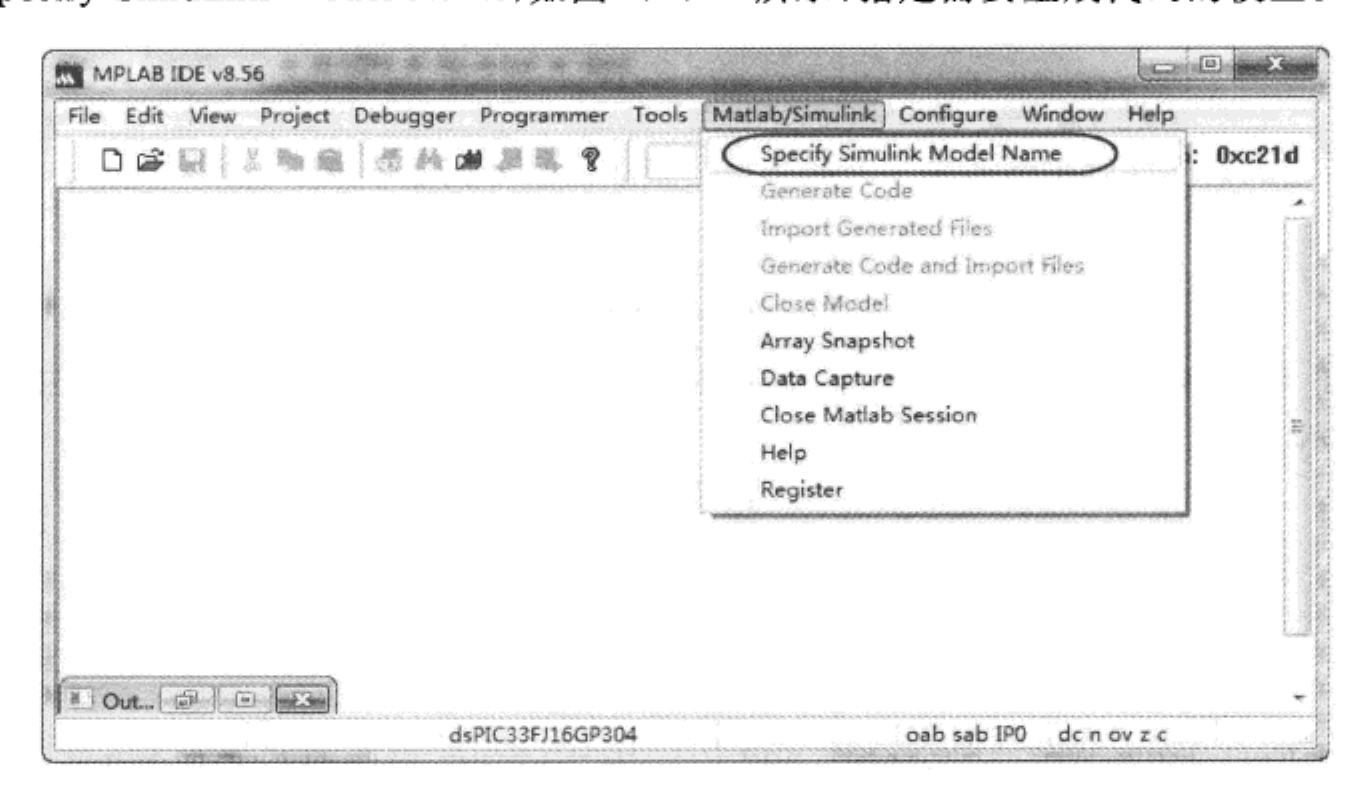


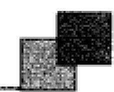
图 7.2.90　指定 Simulink 模型

指定模型并单击确定，如图 7.2.91 所示，系统会自动打开 MATLAB 主窗口与模型窗口（为避免过多占用系统内存，用户应在使用该方法前，关闭所有 MATLAB 窗口），MPLAB IDE 的 Output 窗口显示模型打开成功，如图 7.2.92 所示。



图 7.2.91　选择模型

图 7.2.92　Output 窗口显示模型打开成功



继续选择 MPLAB IDE 的菜单项 Matlab/Simulink→Generate Code，如图 7.2.93 所示。

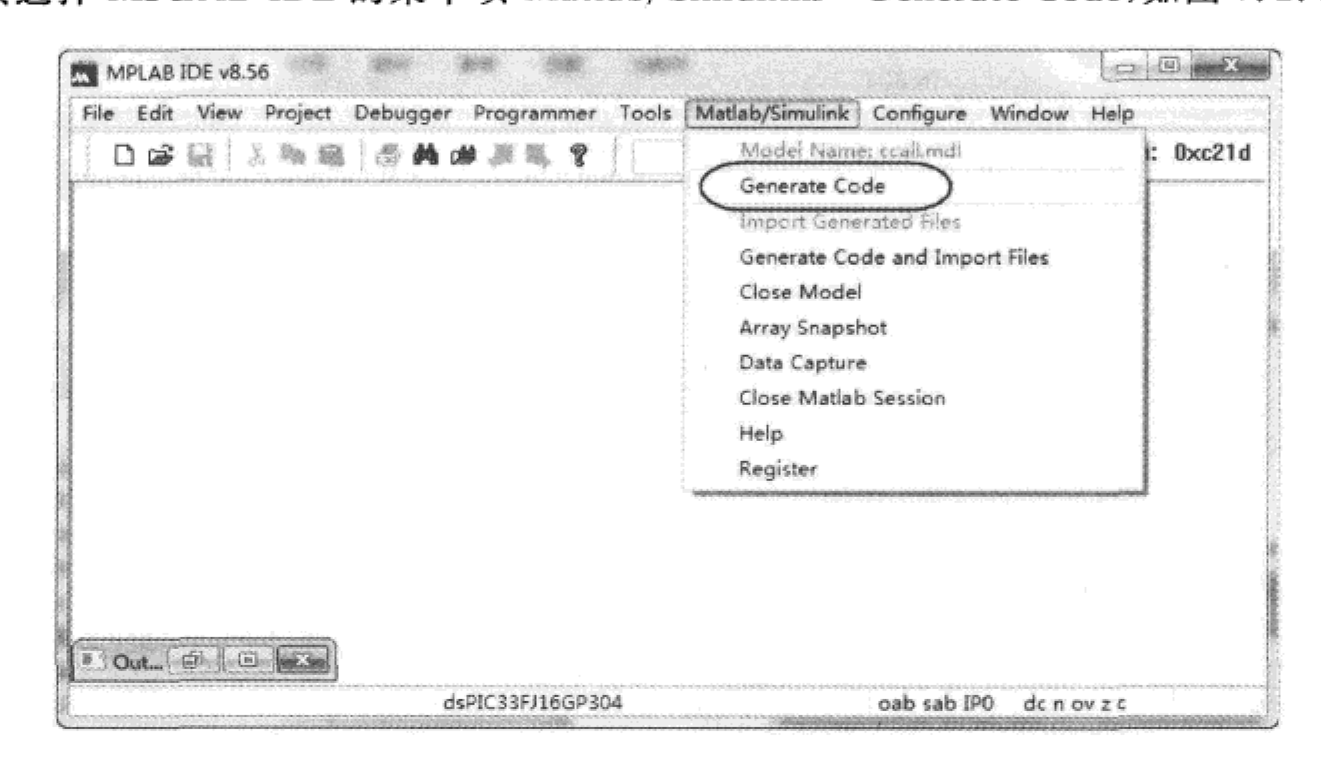


图 7.2.93　选择 Generate Code 命令

等待一会后，系统自动打开 MATLAB 的代码生成报告，如图 7.2.94 所示。

Back　Forward

Contents

Summary
Subsystem Report
Code Interface Report
Traceability Report

Generated Files

[-] **Model files**
ccall.c
ccall.h
ccall_private.h
ccall_types.h

[-] **Utility files**
rtwtypes.h

[-] **Interface files**
autobuild.h

[-] **Other files**
Interrupts.c
main.c

```
12
13  #include "ccall.h"
14  #include "ccall_private.h"
15  #include <dec.c>
16
17  _FWDT(FWDTEN_OFF)
18    _FOSC(FCKSM_CSDCMD&OSCIOFNC_ON&POSCMD_HS);
19  _FOSCSEL(FNOSC_LPRC&IESO_ON);
20  _FGS(GSS_OFF&GCP_OFF&GWRP_OFF);
21  _FICD(ICS_PGD1);
22  void rt_OneStep()
23  {
24    /* Disable interrupts here */
25
26    /* Save FPU context here (if necessary) */
27    /* Re-enable timer or interrupt here */
28    ccall_step();
29
30    /* Get model outputs here */
31
32    /* Disable interrupts here */
33    /* Restore FPU context here (if necessary) */
34    /* Enable interrupts here */
35  }
36
37  int16_T main(void)
38  {
39    PLLFBD = 0;                          // M  = 2
40    CLKDIVbits.PLLPOST = 0;              // N2 = 2
41    CLKDIVbits.PLLPRE = 0;               // N1 = 2
```

图 7.2.94　代码生成报告

之后 MPLAB IDE 的 Output 窗口显示图 7.2.95 所示的信息，说明代码编译成功，但该代码不能直接使用，还需要必要的修改。

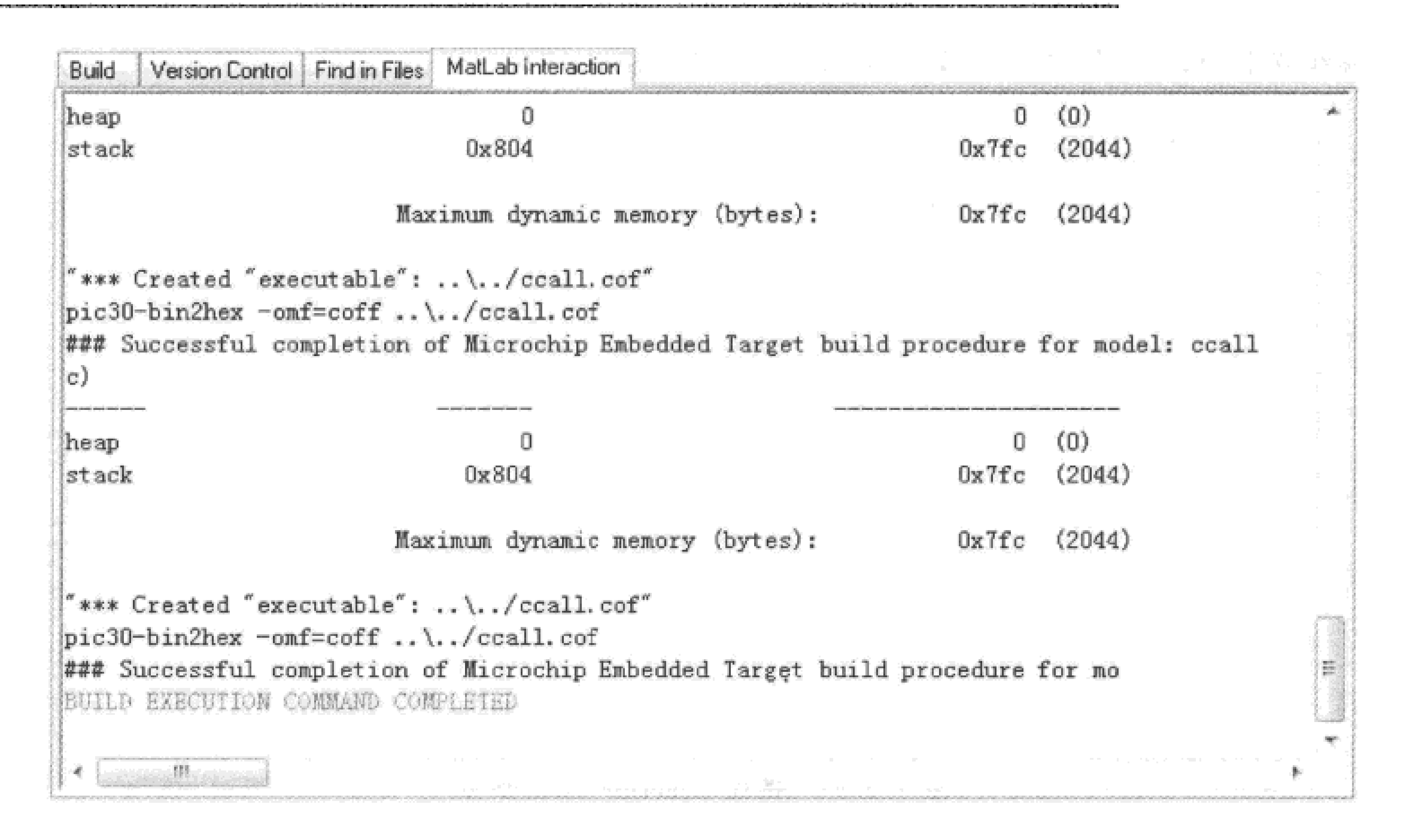


图 7.2.95　Output 窗口信息

在 MPLAB IDE 环境下，新建基于 dsPIC33FJ16GP304 的工程，并加入先前生成的各代码文件以及调用的 C 代码 dec.c，如图 7.2.96 所示。

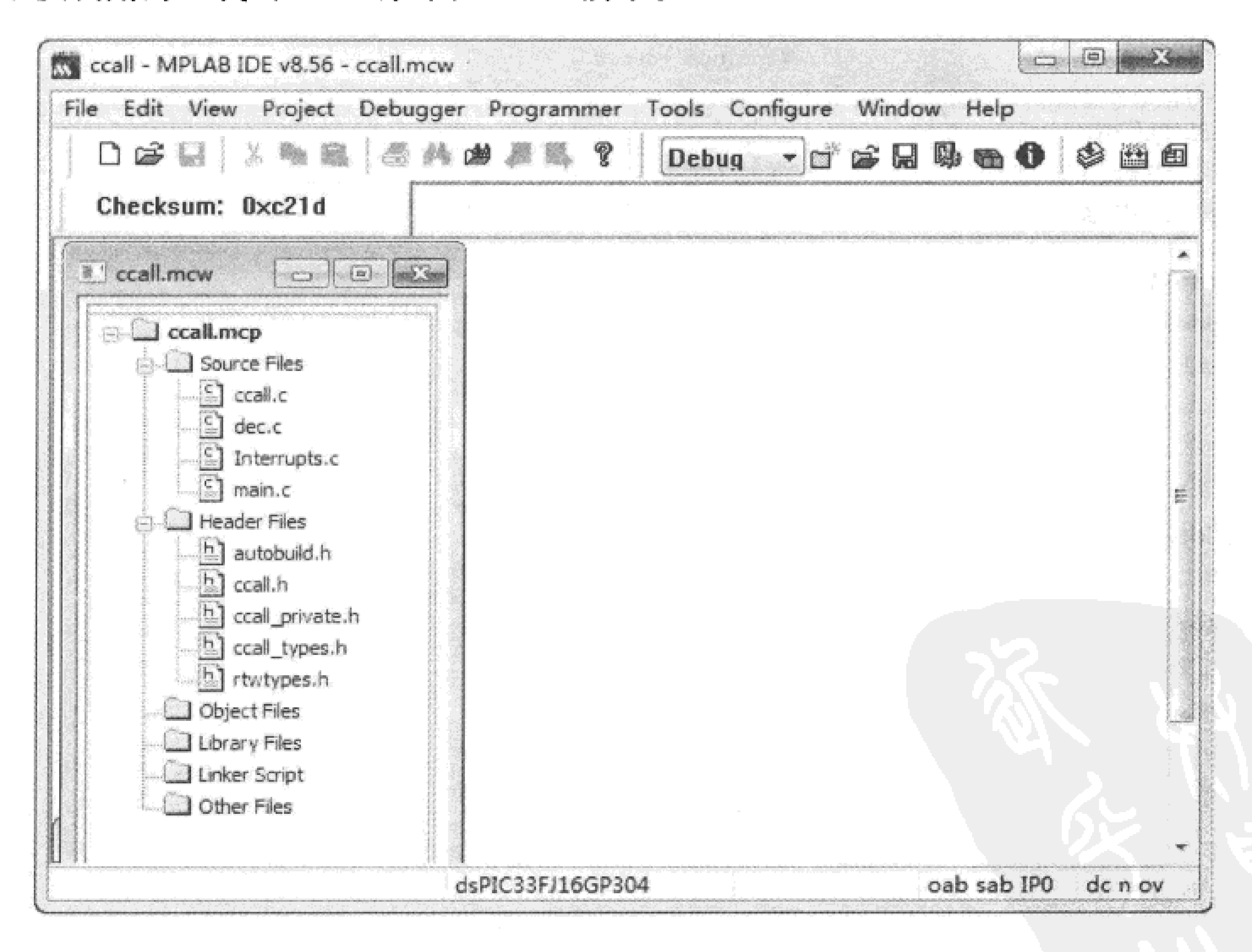
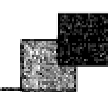


图 7.2.96　建立 MPLAB 工程

打开 main.c，找到以下代码并按注释说明修改：



```
……
#include "ccall.h"
#include "ccall_private.h"
//#include <dec.c>                    //删除该行
……
```

打开 ccall.c,找到以下代码并按注释说明修改:

```
……
#include "ccall.h"
#include "ccall_private.h"
extern int dec();                //增加该行,用于外部函数声明
……
/* S-Function (dsPIC_portRead_sfun): '<Root>/Read Port Input A' */
//rtb_cCall = PORTA;             //删除该行
rtb_cCall = PORTA;               //修改为 rtb_cCall = PORTA>>7;
……
```

保存所有修改,再次编译工程,Output 窗口得到图 7.2.97 所示的信息,说明编译成功。

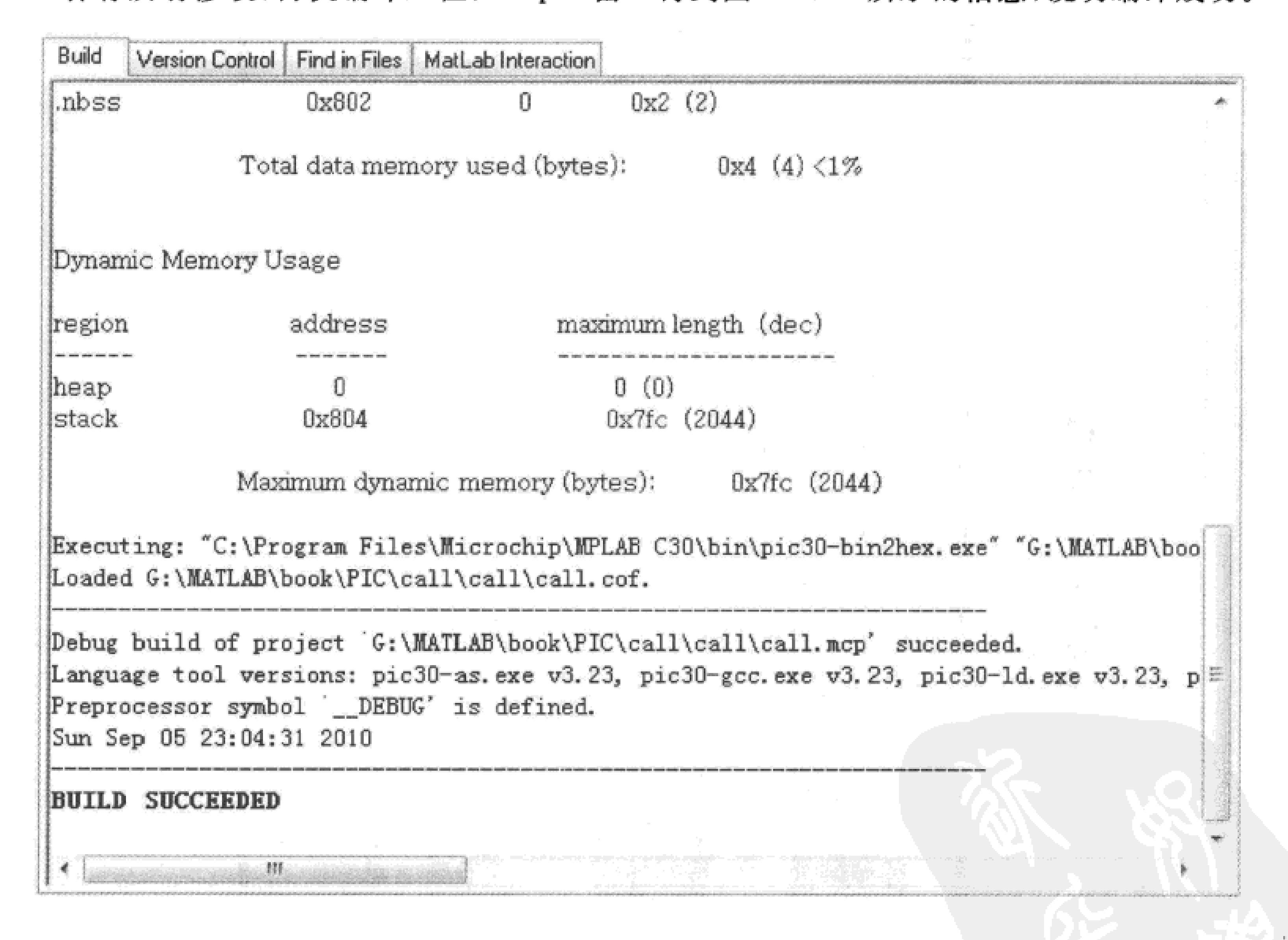


图 7.2.97　编译信息

5. 虚拟硬件测试

搭建 proteus 模型,加载生成的 HEX 文件。执行仿真后,即得到与功能验证模型一致的结果,如图 7.2.98 所示。

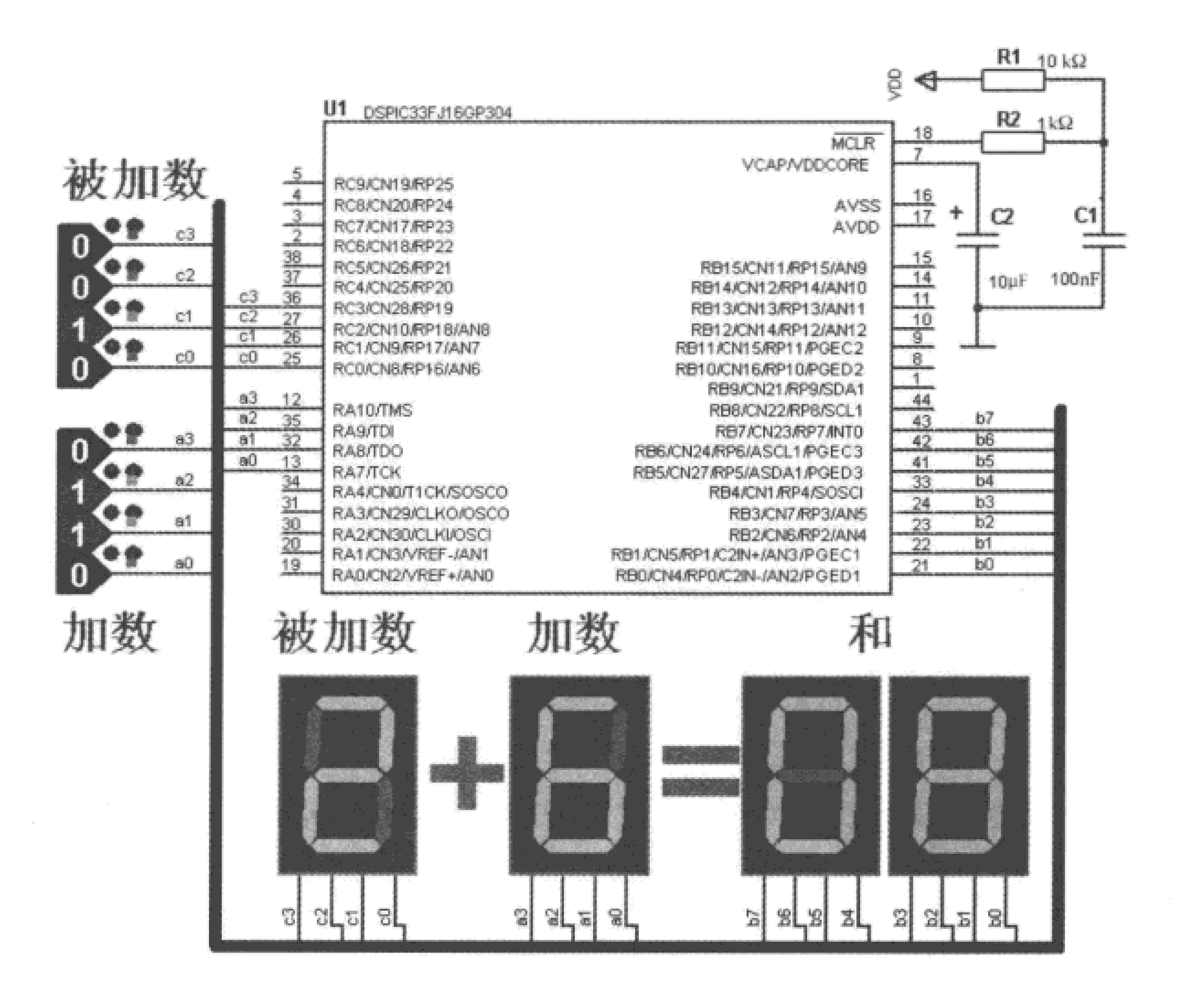


图 7.2.98 仿真结果

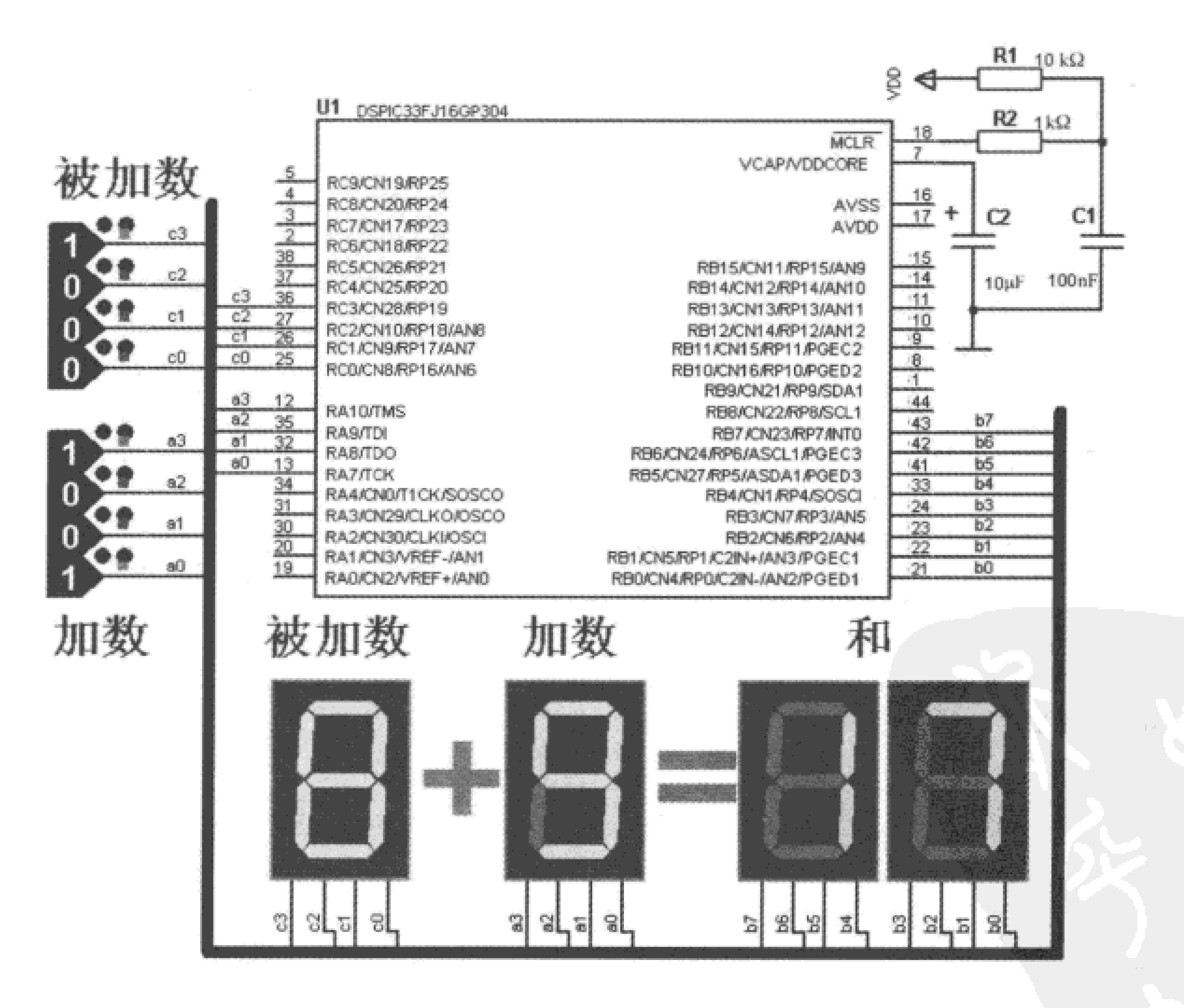
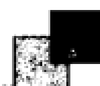


图 7.2.99 仿真结果



7.3　无对应模块时的应用

Microchip 公司提供的 dsPICBlocksets 囊括了 dsPIC 芯片的所有外设，用户可以方便地利用这些模块进行开发工作，摒弃旧的开发方法。但是考虑到 Blocksets 并非免费，会有部分用户不能使用这款软件，也无法使用 dsPIC_stf. tlc 模板，这是否意味着就不能用基于模型的方法进行开发呢？

其实不必担心，用户还可以使用 RTW 提供的 ert. tlc 模板。这样，开发工作中只是外围设备的驱动程序需要用户手工编写，而最重要的核心算法仍然可以用基于模型的方法完成，但代价是代码的效率会有所降低。

7.3.1　创建功能验证模型

图 7.3.1 所示的乘法模型完成了简单的两数相乘，并将其百、十、个位分别显示的功能，根据第 2 章的内容，容易建立此模型。其中 c 表示百位，b 表示十位，a 表示个位。

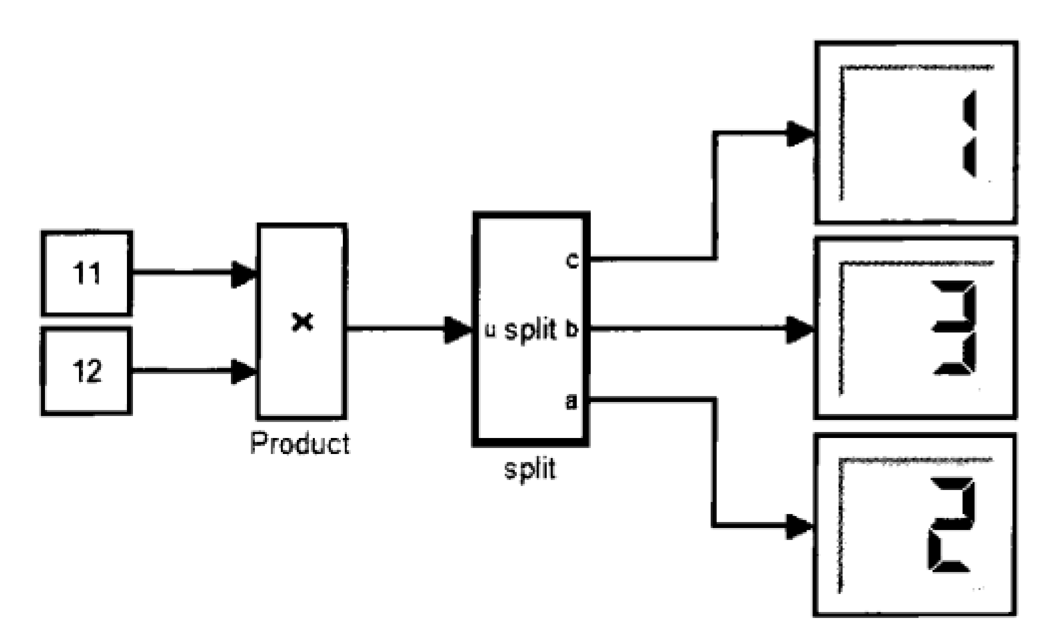


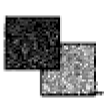
图 7.3.1　功能验证模型

其中 split 是 Embedded MATLAB 模块，完成分离百、十、个位的功能。

```
function [c,b,a] = split(u)
% #eml
a = rem(u,10);                //计算个位
b = (rem(u,100) - a) / 10;    //计算十位
c = (u - 10 * b - a) / 100;   //计算百位
```

7.3.2　自动代码生成

在模型中添加模块库 Simulink / Logic And Bit Operations 中的 Shift Arithmetic 和 Bitwise Operator，在模块库 Simulink / Ports &Subsystems 中找到输入模块与输出模块，替换 Const 和 Generic LED 模块，如图 7.3.2 所示，并将这些端口和 Product 模块的数据类型修改为 uint16。



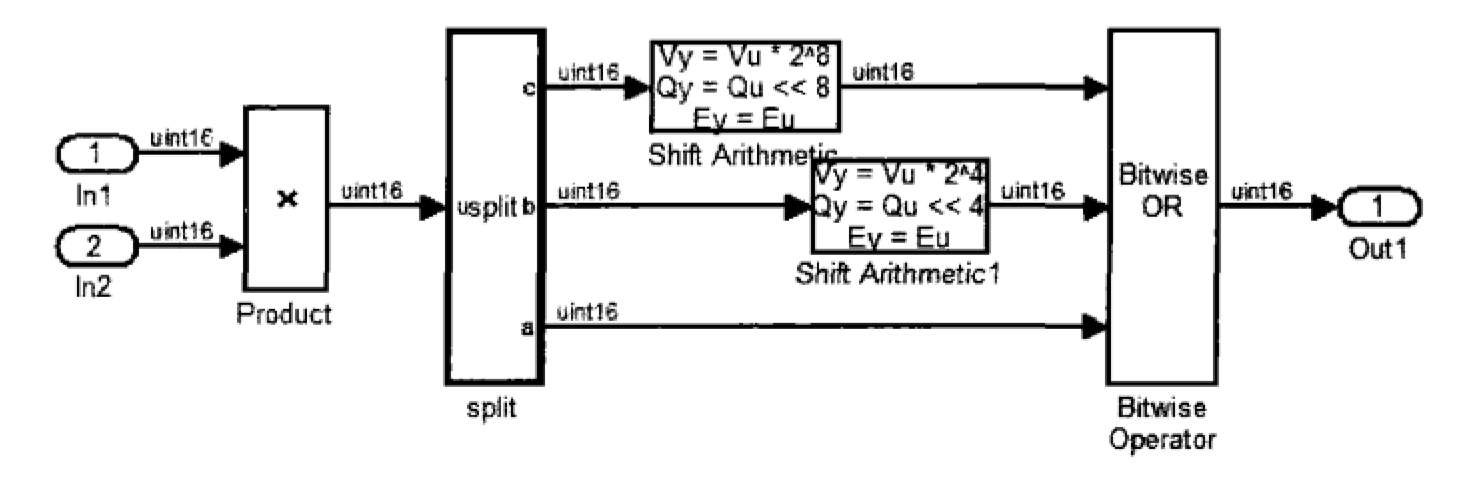


图 7.3.2 代码生成模型

打开 c 端口处的 Shift Arithmetic 模块，在 Number of bits to shift right 中输入－8，将百位数字向左移 8 位，如图 7.3.3 所示。

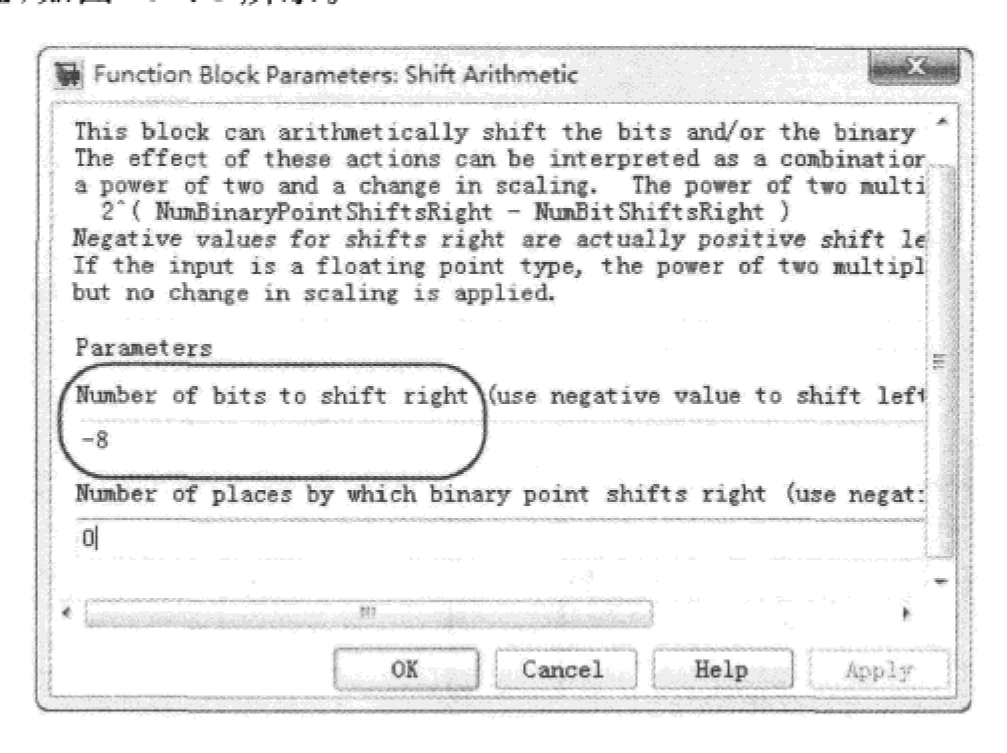


图 7.3.3 算术移位模块设置

打开 b 端口处的 Shift Arithmetic 模块，在 Number of bits to shift right 中输入－4，将十位数字向左移 4 位，如图 7.3.4 所示。

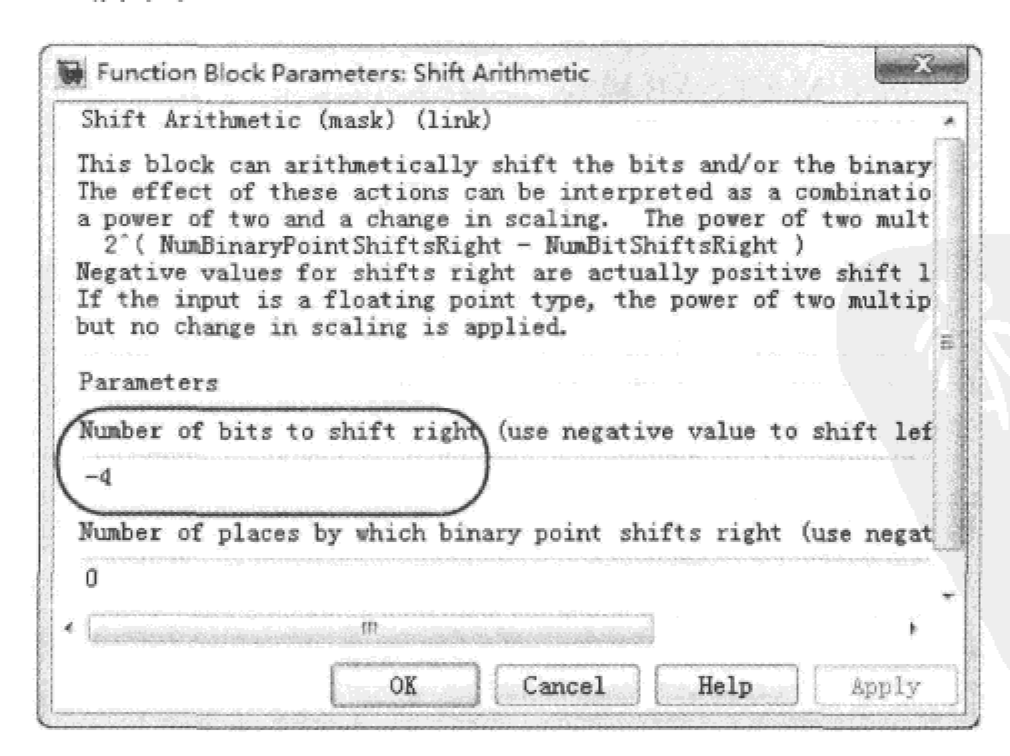
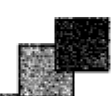


图 7.3.4 算术移位模块设置



打开 Bitwise Operator 模块，设置为或运算(or)，禁用 Use bit mask 功能，将输入端口个数设置为 3 个，如图 7.3.5 所示。

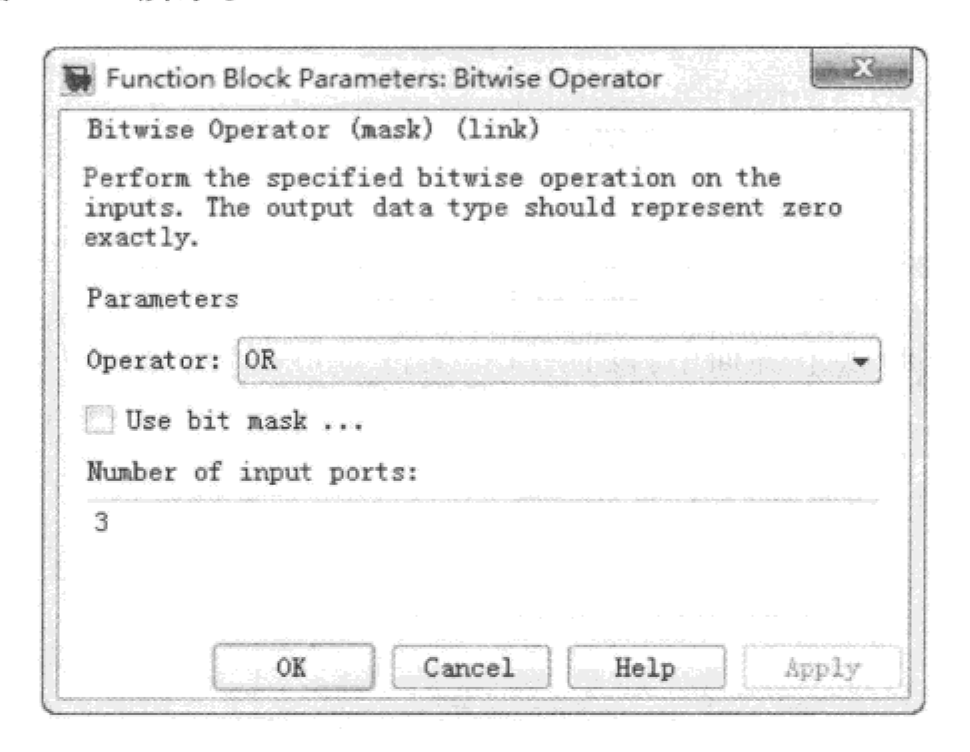


图 7.3.5　Bitwise Operator 模块设置

使用 Shift Arithmetic 和 Bitwise Operator 模块的目的在于将 3 个数据集中到一个端口上输出，提高硬件资源的利用率。计算结果的百位位于寄存器的第 11～8 位，十位位于寄存器的 7～4 位，个位位于寄存器的 3～0 位。

打开模型的参数设置对话框，在 Solver 界面中，设置求解器为定步长离散求解器，如图 7.3.6 所示。

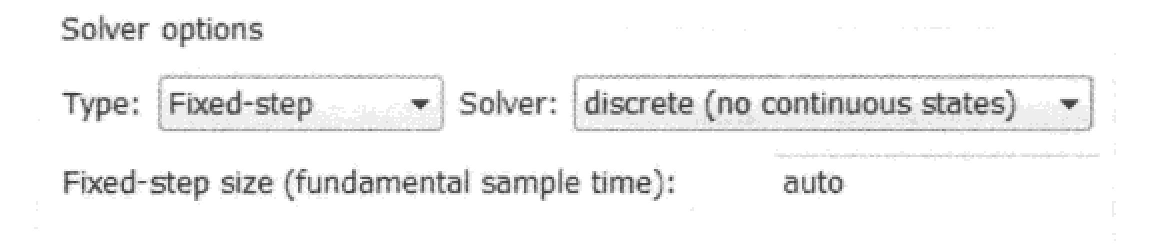


图 7.3.6　求解器设置

在 Report 界面中，勾选所有复选框，便于后期检查及跟踪如图 7.3.7 所示。

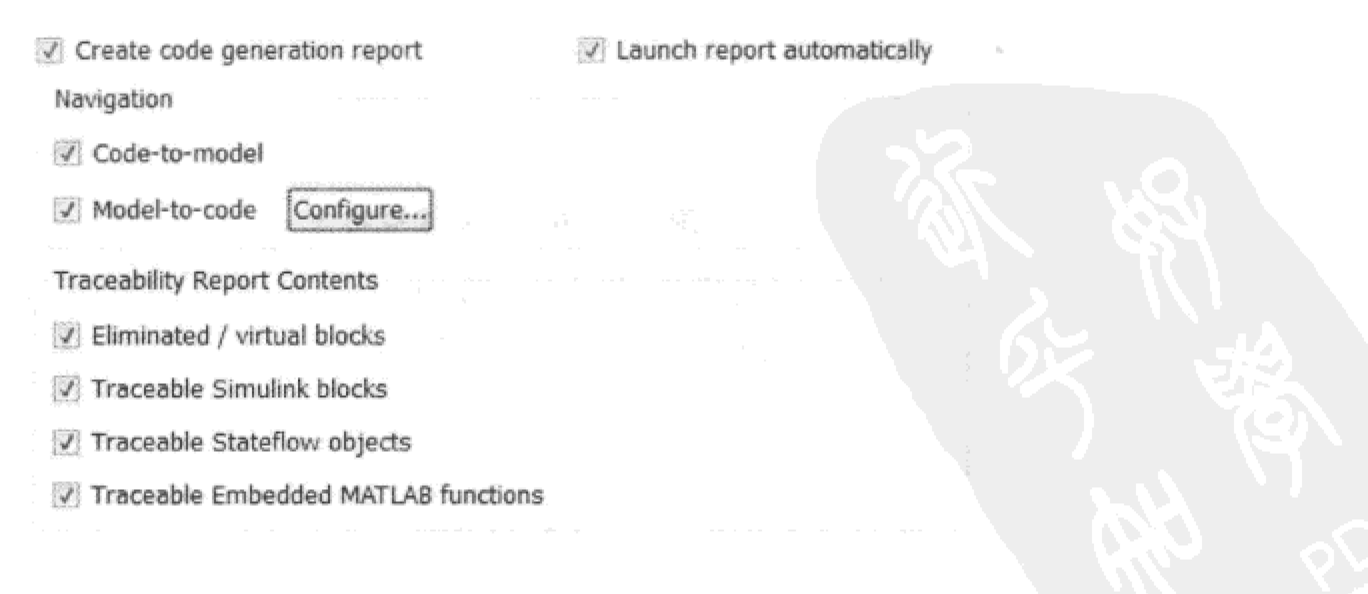


图 7.3.7　报告界面设置

打开模型的参数设置对话框，在 Hardware Implimentation 界面中，设置器件类型为 dsPIC，如图 7.3.8 所示。

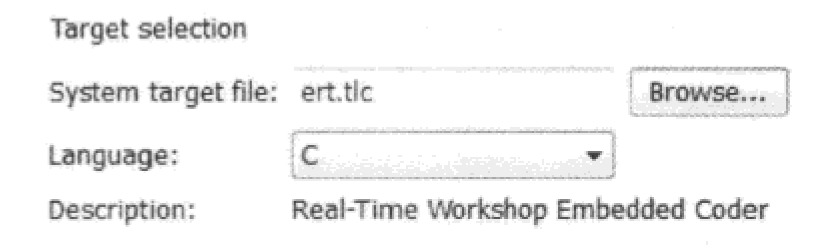


图 7.3.8 选择芯片

在 Real-Time Workshop 界面中，设置 TLC 文件为 ert. tlc，如图 7.3.9 所示。

Target selection
System target file: ert.tlc Browse...
Language: C
Description: Real-Time Workshop Embedded Coder

图 7.3.9 设置 TLC 文件

单击模型工具栏的按钮，生成代码的报告如图 7.3.10 所示。

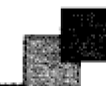
Back Forward
Contents
Summary
Subsystem Report
Code Interface Report
Traceability Report
Generated Files
[-] Main file
ert_main.c
[-] Model files
times_code.c
times_code.h
times_code_private.h
times_code_types.h
[-] Data files
times_code_data.c
[+] Utility files (1)

```
26  * by the base rate from inside the generated code.  Enabling/disabling
27  * interrupts and floating point context switches are target specific.  This
28  * example code indicates where these should take place relative to executing
29  * the generated code step function.  Overrun behavior should be tailored to
30  * your application needs.  This example simply sets an error status in the
31  * real-time model and returns from rt_OneStep.
32  */
33 void rt_OneStep(void);
34 void rt_OneStep(void)
35 {
36   static boolean_T OverrunFlag = 0;
37
38   /* Disable interrupts here */
39
40   /* Check for overrun */
41   if (OverrunFlag) {
42     rtmSetErrorStatus(times_code_M, "Overrun");
43     return;
44   }
45
46   OverrunFlag = TRUE;
47
48   /* Save FPU context here (if necessary) */
49   /* Re-enable timer or interrupt here */
50   /* Set model inputs here */
51
52   /* Step the model */
53   times_code_step();
54
55   /* Get model outputs here */
56
57   /* Indicate task complete */
58   OverrunFlag = FALSE;
59
60   /* Disable interrupts here */
61   /* Restore FPU context here (if necessary) */
62   /* Enable interrupts here */
```

图 7.3.10 代码生成报告

在 MPLAB 中建立工程，将自动生成的代码添加到工程中，由于使用的是 ert. tlc 模板，代码并不能直接使用，需要在 ert_main. c 中作如下修改：

```
……
//#include <stdio.h>                /* 去掉该行 */
#include "times_code.h"             /* 模型头文件 */
#include "rtwtypes.h"               /* MathWorks 数据类型 */
#include "p33FJ16GP304.h"           //添加 dsPIC33FJ16GP304 头文件
```



```
......
    /* Save FPU context here (if necessary) */
    /* Re-enable timer or interrupt here */
    /* Set model inputs here */
    times_code_U.In1 = PORTC;           //模型输入口与硬件相关联
    times_code_U.In2 = PORTA >> 7;      //模型输入口与硬件相关联
    /* Step the model */
    times_code_step();
    /* Get model outputs here */
    PORTB = times_code_Y.Out1;          //模型输出口与硬件相关联
......
int_T main();                           //删除不必要的输入参数
int_T main()                            //删除不必要的输入参数
{
  /* Initialize model */
  times_code_initialize();
    TRISA = 0x7ff;                      //指定 A 端口为输入
    TRISB = 0;                          //指定 B 端口为输出
    TRISC = 0x3ff;                      //指定 C 端口为输入
while(1)
{rt_OneStep();}
......
  //删除下列代码
  // printf("Warning: The simulation will run forever. "
  //   "Generated ERT main won't simulate model step behavior. "
    //   "To change this behavior select the 'MAT-file logging' option.\n");
  // fflush((NULL));
......
}
```

保存所有修改，再次编译工程，Output 窗口得到以下信息，说明编译成功(图 7.3.11)。

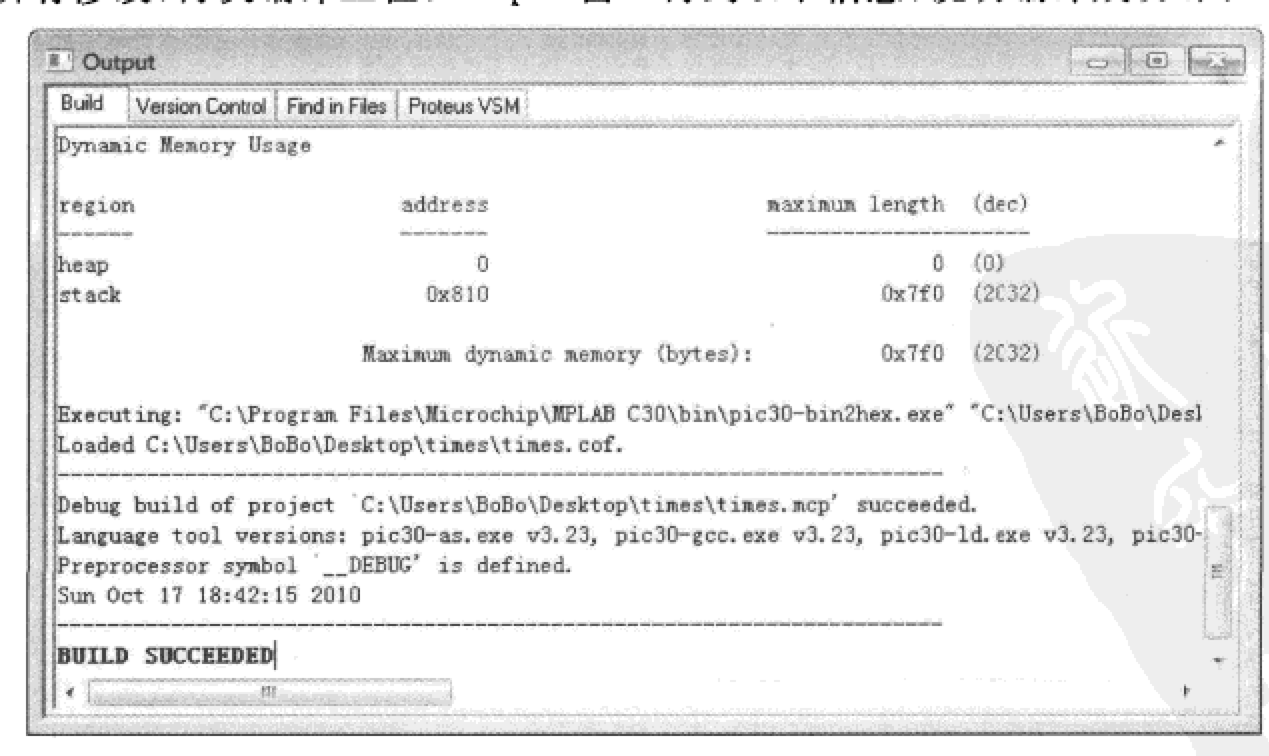


图 7.3.11　编译信息

7.3.3 虚拟硬件测试

本例利用 dsPIC33FJ16GP304 芯片实现乘法的输入与显示，参考第 5.1 节的内容，搭建出图 7.3.12 所示的原理图。

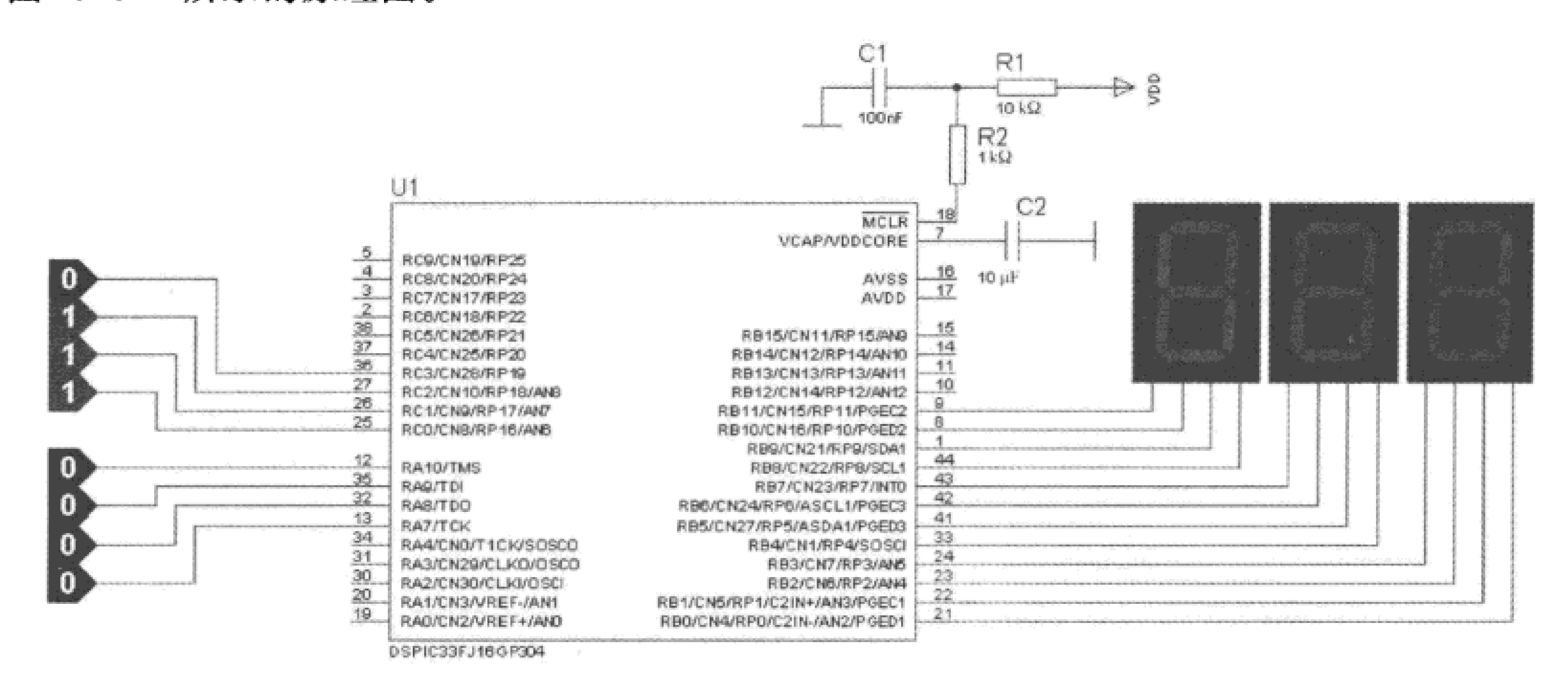


图 7.3.12 Proteus 原理图

第一个输入数据从 RC0～RC3 引脚进入芯片，第二个数据从 RA7～RA10 引脚进入芯片。由十进制与二进制的关系可知，输入数值的范围为 0～15。

数码管的百位和 RB8～RB11 引脚连接，十位和 RB4～RB7 引脚连接，个位和 RB0～RB3 引脚连接。

重新编译后，利用 Proteus VSM 进行调试，输入数据为 1001、1101，对应的十进制数字为 9、13，可以看到输出结果显示为 117，实现了模型的预期功能，如图 7.3.13 所示。

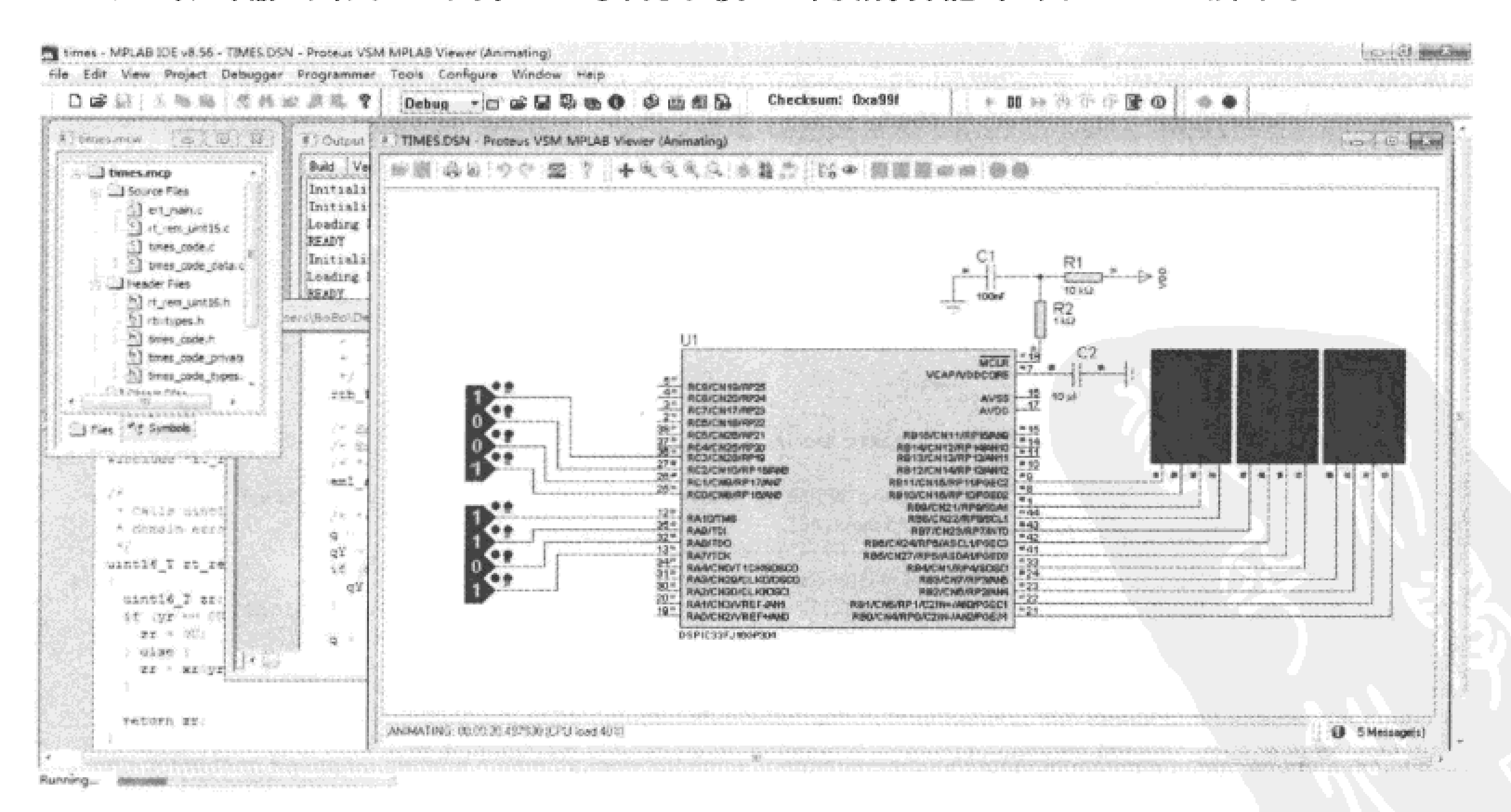


图 7.3.13 仿真结果

第 8 章

ARM 代码的快速生成

开发 ARM 处理器程序的方法有两种:有操作系统的程序设计;无操作系统的程序设计,即裸机——单片机开发方式。MathWorks 公司在 MATLAB R2010a 就开始引入基于 Eclipse 的开发和 Linux 操作系统,支持除 Cortex 系统以外的所有 ARM 芯片。在刚发布的 MATLABR2010b 版中,不但引入了 Vxworks 操作系统,还增加了对 ARM Cortex 系列芯片的支持。本章仅讨论无操作系统的基于模型设计的 ARM 开发。考虑到有些读者可能没有具体的硬件设备,这里仅就 Porteus 虚拟硬件平台支持的 NXP ARM7 进行介绍,其他型号 ARM 芯片的裸机开发与此完全类似。还有,本章所举的实例仅仅是为了增加读者的观赏性,ARM 处理器的应用领域远不止这些,对于其他领域的应用,读者可根据实际情况借鉴本章的思想进行。对于目前流行的低功耗、高效率的新一代 ARM 芯片 Cortex 系列,在作者的后续书中将专门讨论。

本章的主要内容:

- ARM 简介。
- 蜂鸣器。
- 交通灯控制。
- 步进电动机控制。
- 无刷电动机控制。

8.1 ARM 简介

ARM 是 Advanced RISC Machines 的缩写,是一家专门授权 IP(知识产权)给全球众多著名半导体、软件和 OEM 厂商,并提供服务的英国公司。其产品包括 ARM7、ARM9、ARM10、ARM11,目前最新型的是 ARM Cortex 系列。下面就本章用到的 ARM7 TDMI-S 芯片作简单介绍。

ARM7 TDMI-S 是通用的 32 位微处理器,它具有高性能和低功耗的特性。ARM 结构是为基于精简指令集原理而设计的。使用一个小的、廉价的微处理器核,便可得到很高的指令吞吐量和实时的中断响应。采用流水线技术,在执行一条指令的同时对下一条指令进行译码,并

将第三条指令从存储器中取出。ARM 处理器同时包含了 ARM 指令集和 THUMB 指令集，THUMB 指令特别适用于存储容量小，代码密度大的场合。LPC2103 是一款基于 ARM7 TDMI-S 体系结构的 ARM7 芯片，它的主要性能如下：

图 8.1.1　LPC2103 结构图

(1) 16/32 位 ARM7 TDMI-S 微控制器，超小 LQFP48 封装。

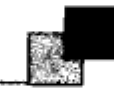
(2) 8 KB 的片内静态 RAM 和 32 KB 的片内 Flash 程序存储器。128 位宽度接口/加速器可实现高达 70 MHz 工作频率。

(3) 通过片内 boot 装载程序实现在系统/在应用编程。单个 Flash 扇区或整片擦除时间为 100 ms。256 字节编程时间为 1 ms。

(4) 嵌入式 ICE RT 通过片内 RealMonitor 软件提供实时调试。

(5) 10 位 A/D 转换器提供 8 路模拟输入，以及特定的结果寄存器来最大限度地减少中断开销。

(6) 2 个 32 位定时器/外部事件计数器(带 7 路捕获和 7 路比较通道)。



(7) 2 个 16 位定时器/外部事件计数器(带 3 路捕获和 7 路比较通道)。

(8) 低功耗实时时钟(RTC)具有独立的电源和特定的 32kHz 时钟输入。

(9) 多个串行接口,包括 2 个 UART、2 个高速 I^2C 总线、SPI 和具有缓冲作用和数据长度可变功能的 SSP。

(10) 向量中断控制器(VIC),可配置优先级和向量地址。

(11) 多达 32 个通用 I/O 口。

(12) 多达 13 个边沿或电平触发的外部中断引脚。

(13) 通过一个可编程的片内 PLL 可实现最大为 70MHz 的 CPU 操作频率,其具有 10～25 MHz 的输入频率。

(14) 片内集成振荡器与外部晶体的操作频率范围为 1～25MHz。

(15) 低功耗模式包括空闲模式、掉电模式和带有效 RTC 的掉电模式。

(16) 可通过个别使能/禁止外围功能和外围时钟分频来优化额外功耗。

(17) 通过外部中断或 RTC 将处理器从掉电模式中唤醒。

读者想了解更多的 LPC2103 微控制器的信息请阅读参考文献或查阅 NXP 官方网站上的数据手册。

8.2　蜂鸣器

8.2.1　蜂鸣器发声模型

蜂鸣器内部的振动膜片在电磁线圈和磁铁的相互作用下,周期性地振动发声。由此可知,脉冲信号即可驱动其电磁线圈,发出蜂鸣声。根据第 3 章的介绍,容易建立蜂鸣器的 Stateflow 模型,如图 8.2.1 所示。

其中数据 io0set、io0clr 表示生成的脉冲信号;pinsel0、io0dir 为芯片端口的状态设置;flag 表示高低电平间的转换条件标志;key 控制蜂鸣器是否工作,如图 8.2.2 所示。

虽然 ARM 系列芯片已经集成了 PWM 功能,但这里为了验证基于模型设计的可行性,此例采用软件方式产生 PWM 信号。

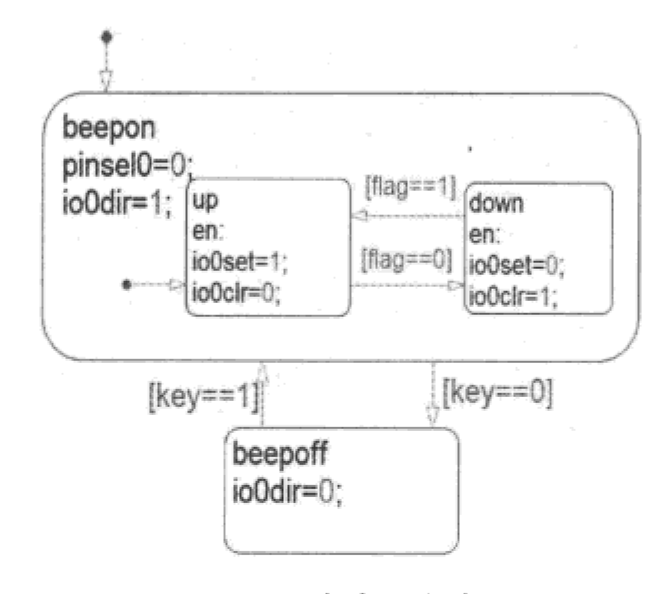
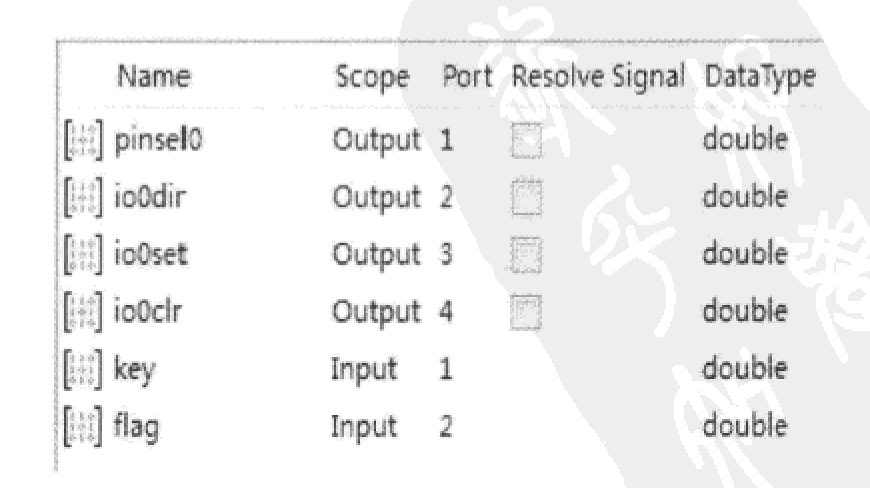


图 8.2.1　蜂鸣器发声模型



Name	Scope	Port	Resolve Signal	DataType
pinsel0	Output	1	☐	double
io0dir	Output	2	☐	double
io0set	Output	3	☐	double
io0clr	Output	4	☐	double
key	Input	1		double
flag	Input	2		double

图 8.2.2　蜂鸣器发声模型数据列表

8.2.2 蜂鸣器功能验证模型

完成蜂鸣器发声模型之后，在 Simulink 模块库中找到图 8.2.3、图 8.2.4 所示模块，并按图 8.2.5 连接。

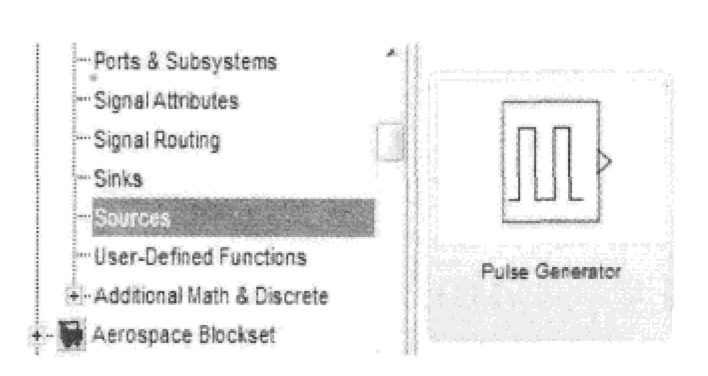


图 8.2.3 脉冲发生器模块　　　　图 8.2.4 示波器模块

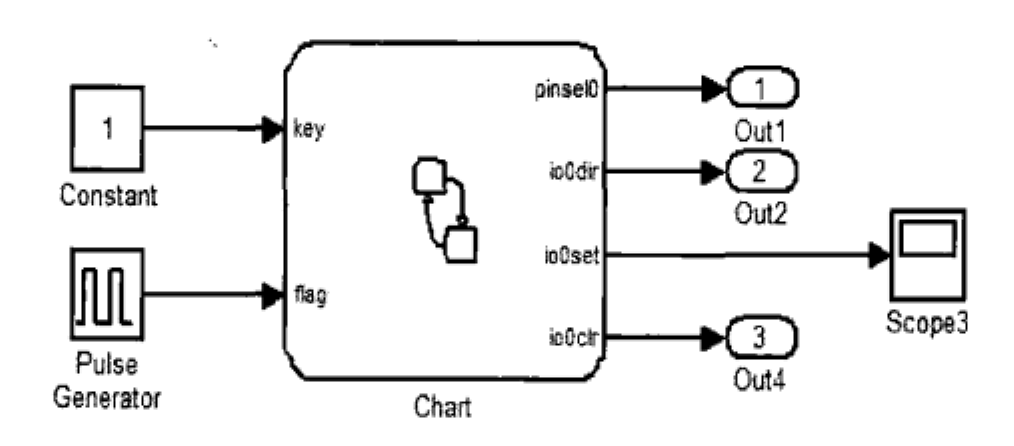


图 8.2.5 功能验证模型

选择模型主窗口的菜单项 Simulation→Configuration Parameters...，打开模型参数对话框，在 Solver 面板，设置求解器为定步长离散求解器，步长为 0.01，如图 8.2.6 所示。

Constant 模块将值设为 1，使模型处于 beepon 状态；Pulse Generator 模块产生脉冲，使 flag 不断变化。

运行模型，可以看到 io0set 端口输出了脉冲信号，如图 8.2.7 所示。

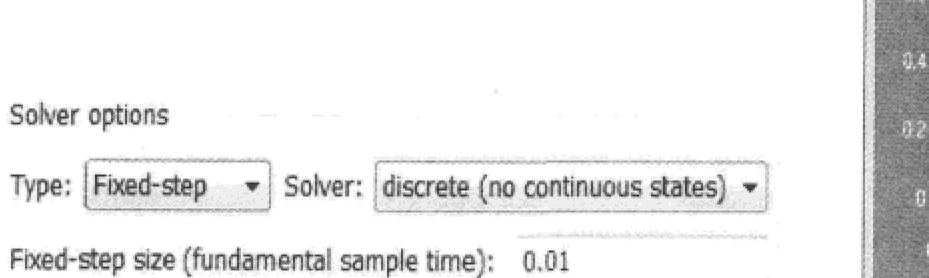


图 8.2.6 求解器设置

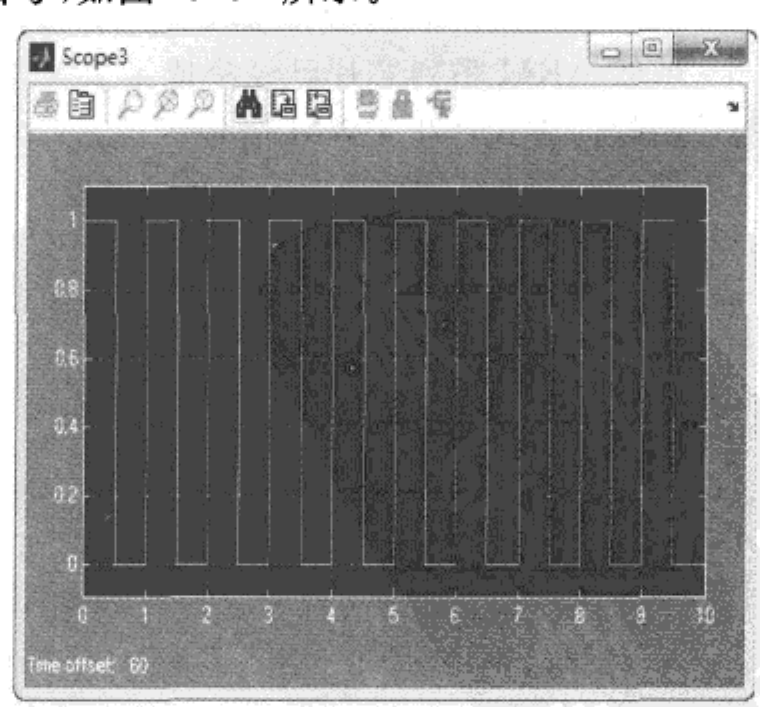
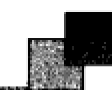


图 8.2.7 仿真结果



8.2.3 软件在环测试

软件在环测试(SIL)是在主机上对仿真中生成的函数或手写代码进行非实时性联合仿真评估,当软件组件包含需要在目标平台上执行的生成代码和手写代码的组合时,应该考虑进行软件在环测试,完成对模型生成代码的早期验证。

软件在环测试不需要硬件,只是对算法代码进行测试,具体做法是对要进行测试的子系统编译可生成 SIL 模块,比较原模块与 SIL 模块的输出,以此确认算法的正确性。

在模块库 Simulink / Ports &Subsystems 中找到输入/输出模块,替换图 8.2.7 中的脉冲信号源与常量输出,并将这些端口的数据类型修改为 uint32,如图 8.2.8 所示。

另外还需要确保 Stateflow 模型中数据的类型也应为 uint32,如图 8.2.9、图 8.2.10 所示。

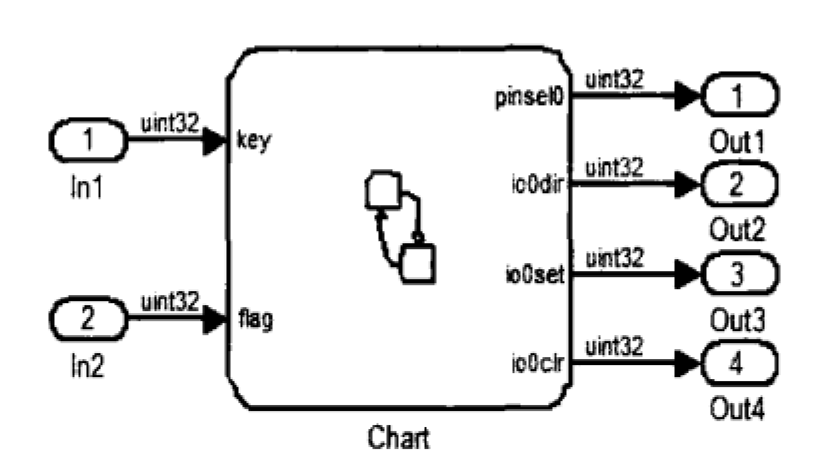


图 8.2.8 代码模型

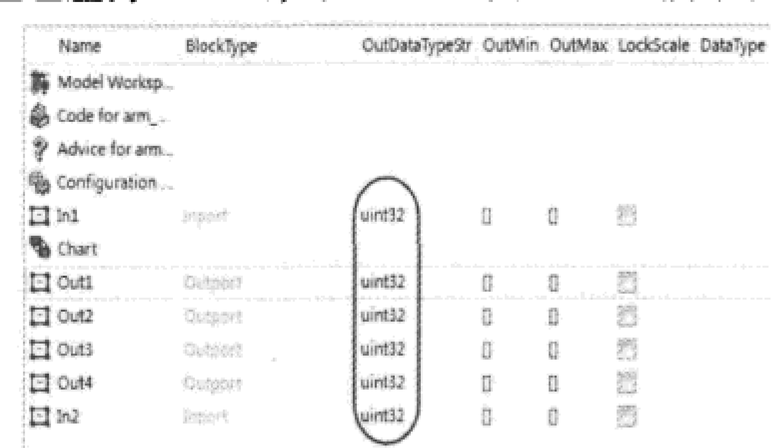


图 8.2.9 修改模型端口数据类型

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	Compile
pinsel0	Out...	1	☐	uint32			uint32	1
io0dir	Out...	2	☐	uint32			uint32	1
io0set	Out...	3	☐	uint32			uint32	1
io0clr	Out...	4	☐	uint32			uint32	1
key	Input	1		uint32			uint32	1
flag	Input	2		uint32			uint32	1

图 8.2.10 修改模型内部数据类型

Report 界面,勾选所有复选框,便于后期检查及跟踪,如图 8.2.11 所示。

在 Real-Time Workshop 界面,设置 TLC 文件为 ert. tlc,如图 8.2.12 所示。

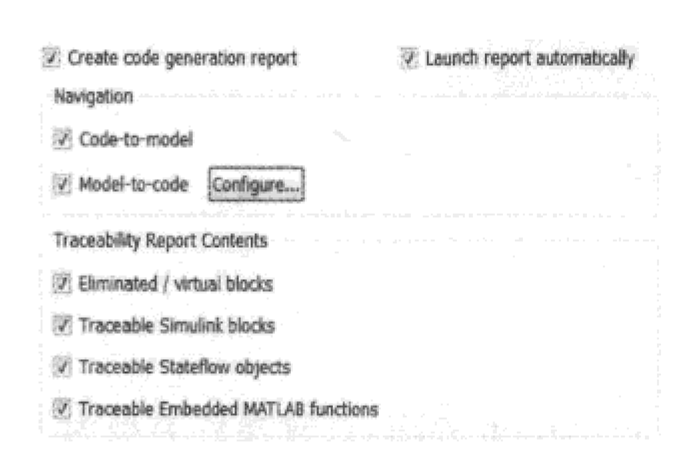


图 8.2.11 报告界面设置

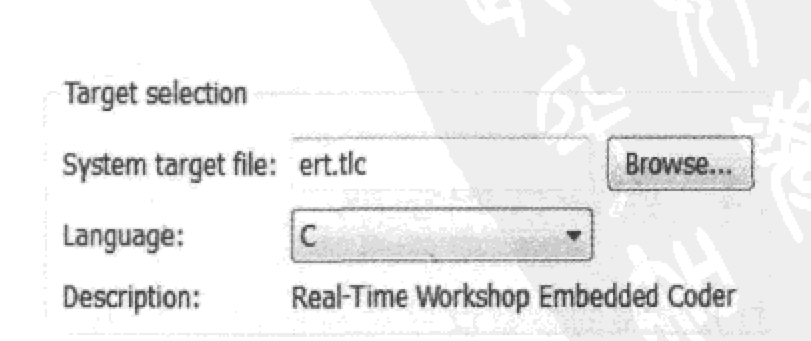


图 8.2.12 设置 tlc

打开模型的参数设置对话框，在 Real-Time Workshop→SIL and PIL Verification 界面，勾选 Create SIL block 复选框(图 8.2.13)。

之后按下模型工具栏的按钮，生成 SIL 模块(图 8.2.14)。

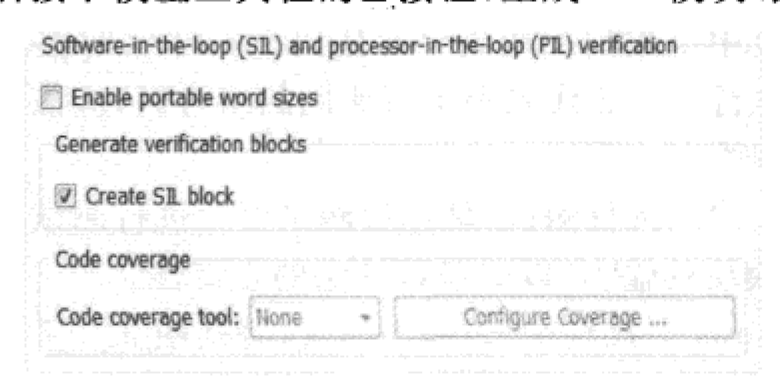


图 8.2.13　SIL 设置

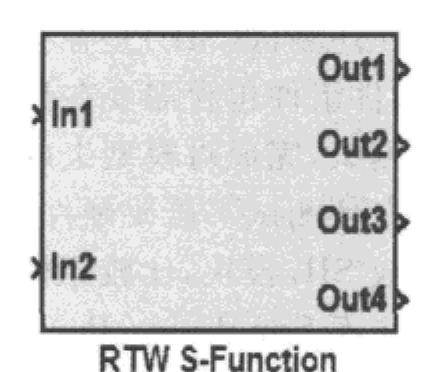


图 8.2.14　SIL 模块

如图 8.2.9 所示，以 SIL 模块替换 Stateflow 模块，重建图 8.2.15 所示的验证模型。由于 SIL 模块是根据定点模型建立的，因此各端口间加入了数据类型转换模块。

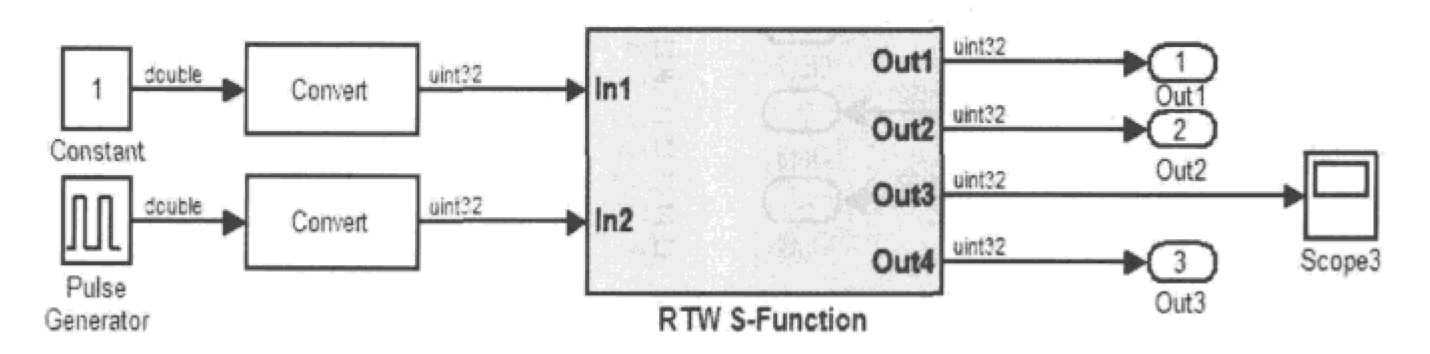


图 8.2.15　SIL 测试模型

该模型的运行结果如图 8.2.16 所示，与 Simulink 功能验证模型的结果一致。

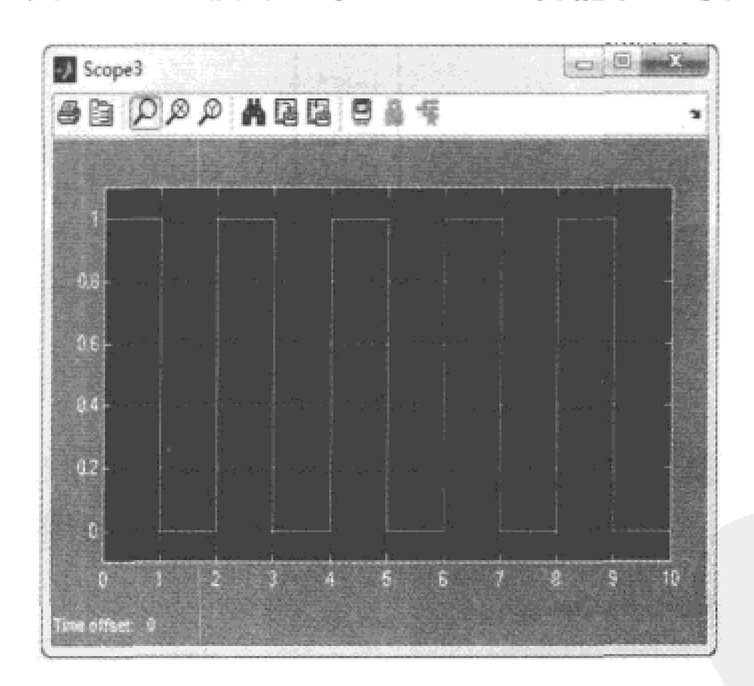
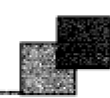


图 8.2.16　仿真结果

8.2.4　自动代码生成

打开模型的参数设置对话框，在 Hardware Implimentation 界面，设置器件类型为 ARM 7，如图 8.2.17 所示。



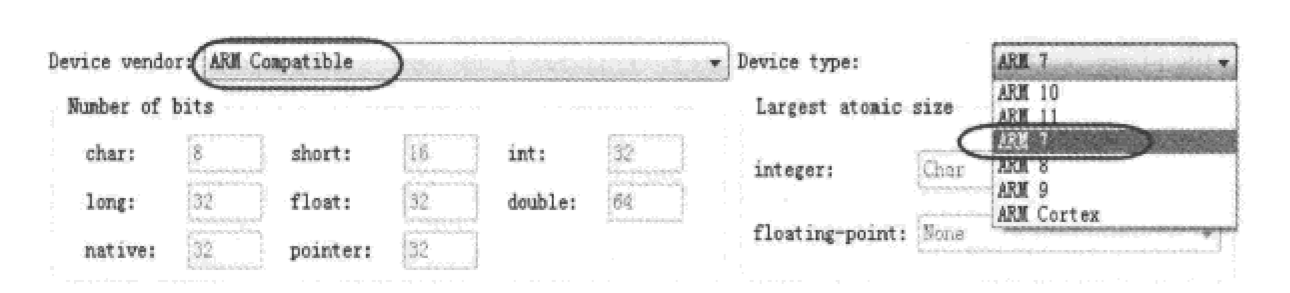


图 8.2.17　选择芯片

单击模型工具栏的按钮，生成代码的报告如图 8.2.18 所示。

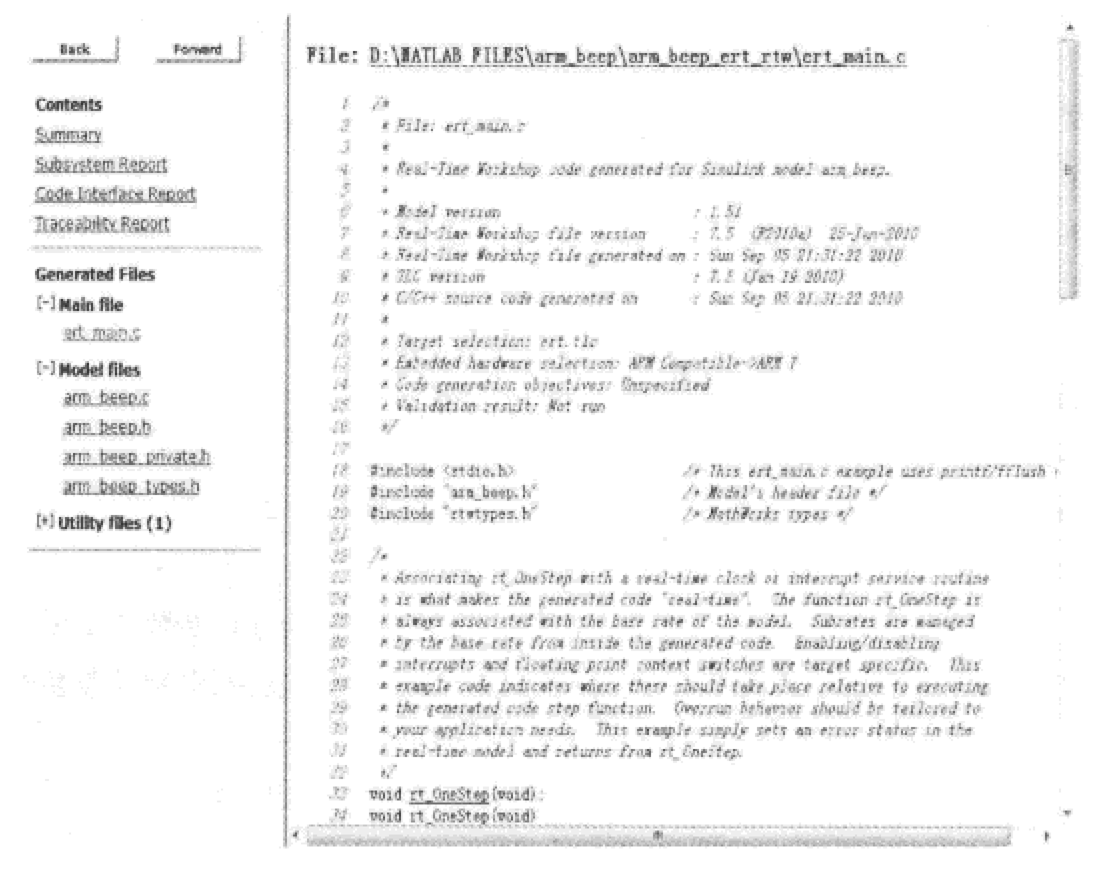


图 8.2.18　代码生成报告

1. 建立 Keil 工程

打开 Keil uVision4，芯片选择 LPC2103，建立工程，并加入生成的代码，如图 8.2.19 所示。

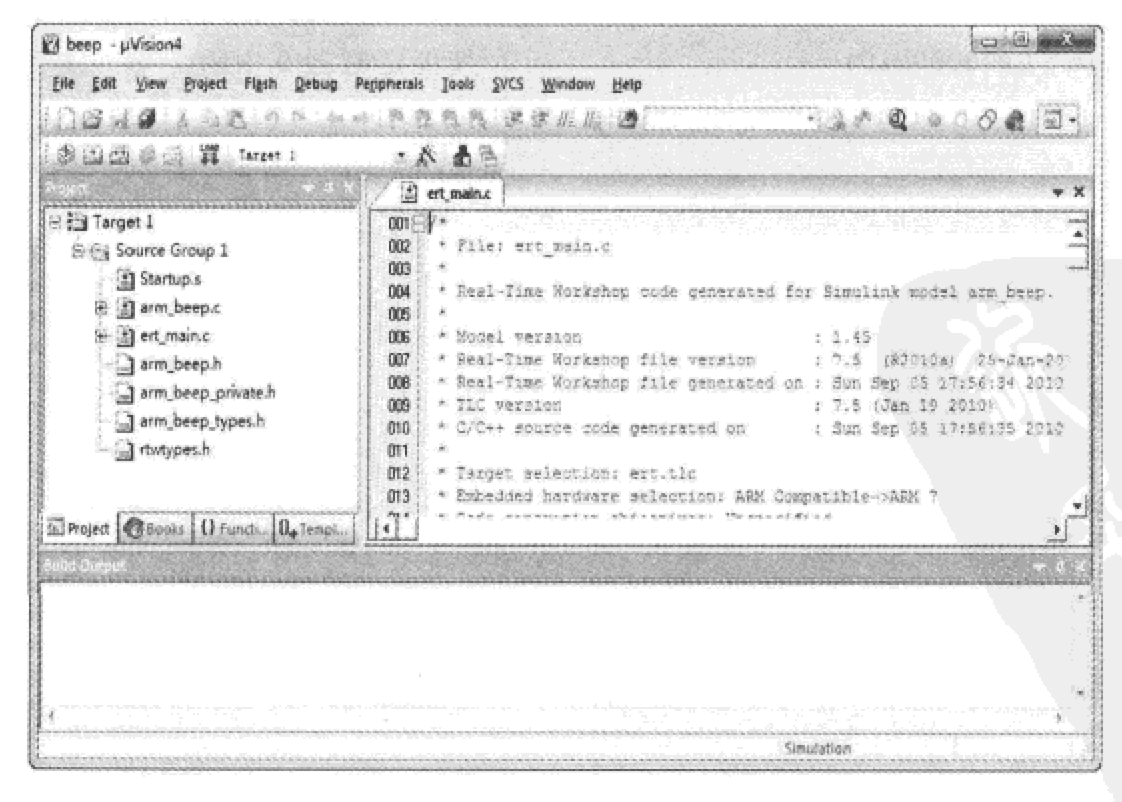


图 8.2.19　Keil 工程

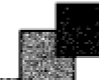
代码并不能直接使用，还需在 ert_main.c 中作如下修改：

```
.........
// #include <stdio.h>  /* 在后面的代码中删除了函数 printf/fflush,因此不再需要 stdio.h */
#include "arm_beep.h"                      /* Model's header file */
#include "rtwtypes.h"                      /* MathWorks types */
#include <LPC21XX.H>                       /* 添加 LPC21XX 头文件 */
/* 手工添加 Timer0 的终端服务程序 */
void __irqflag(void)
{
  if(arm_beep_U.In2 == 0) arm_beep_U.In2 = 1;   //flag 置 1
     else arm_beep_U.In2 = 0;                   //flag 置 0
     T0IR = 0x01;                               //清除中断标志
     VICVectAddr = 0x00;                        //通知 VIC 中断处理结束
     }
/* 手工添加 Timer0 的初始化代码 */

void Timer0Init(void)                          //初始化定时器 0
{
  T0PR = 99;                                   //设置定时器分频系数为 100
  T0MCR = 0x03;                                //匹配通道 0 匹配中断并复位 T0TC
  T0MR0 = 120/2;                               //比较值
  T0TCR = 0x03;                                //启动并复位 T0TC
  T0TCR = 0x01;
  /* 设置定时器 0 中断 IRQ */
  VICIntSelect = 0x00;                         //所有中断通道设置为 IRQ 中断
  VICVectCntl0 = 0x24;                         //定时器 0 中断设为最高优先级
  VICVectAddr0 = (uint32_T)flag;               //设置中断服务程序地址向量
  VICIntEnable = 0x00000010;                   //使能定时器 0 中断

}
............
.........
/* Set model inputs here */
    /* 将模型输入与硬件相对应 */
    /* Step the model */
    arm_beep_step();

    /* Get model outputs here */
/* 将模型输出与硬件相对应 */
........
int_T main(int_T argc, const char_T *argv[]);
```



```
int_T main(int_T argc, const char_T *argv[])
{
  /* Initialize model */
  arm_beep_initialize();
  TimerOInit();                          //调用定时器 0 初始化函数

/*删除下列代码*/
// printf("Warning: The simulation will run forever. "
//         "Generated ERT main won't simulate model step behavior. "
//         "To change this behavior select the 'MAT-file logging' //option.\n");
//  fflush((NULL));
  while (rtmGetErrorStatus(arm_beep_M) == (NULL)) {
/* Perform other application tasks here */
     rt_OneStep();                       //调用 rt_OneStep
  }

  /* Disable rt_OneStep() here */

  /* Terminate model */
  arm_beep_terminate();
  return 0;
}
........
```

2. 生成 hex 文件

单击 Keil 工具栏的按钮，或按 Alt+F7 组合键，在 Output 选项卡中，点选 Create HEX File 单选按钮，如图 8.2.20 所示。

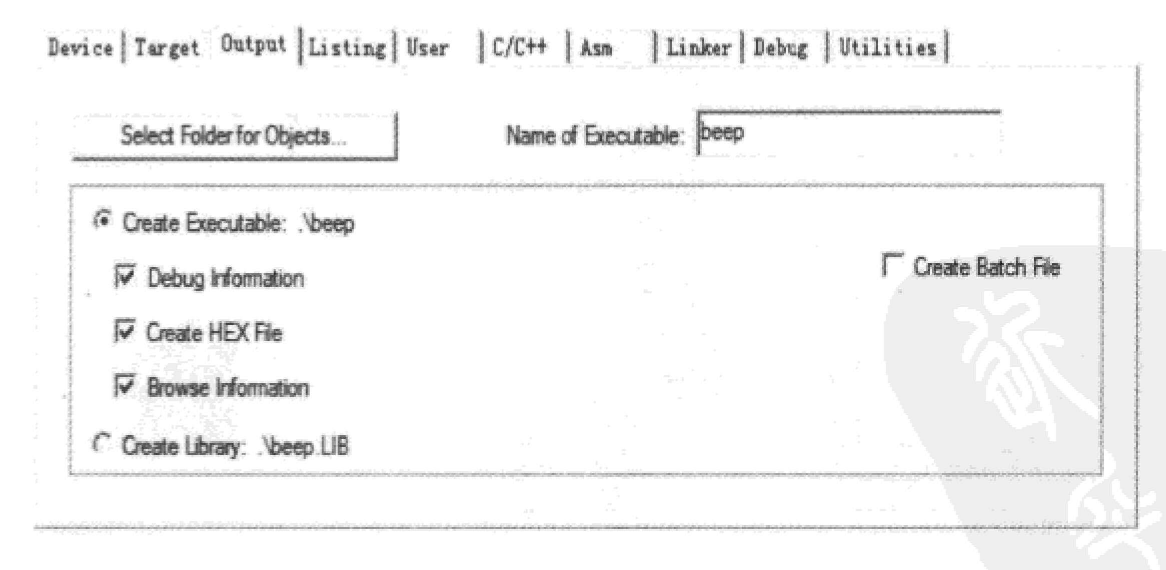


图 8.2.20　设置输出.hex 文件

再单击工具栏按钮，重编译工程(图 8.2.21)，窗口下部的信息显示已成功生成.hex 文件。

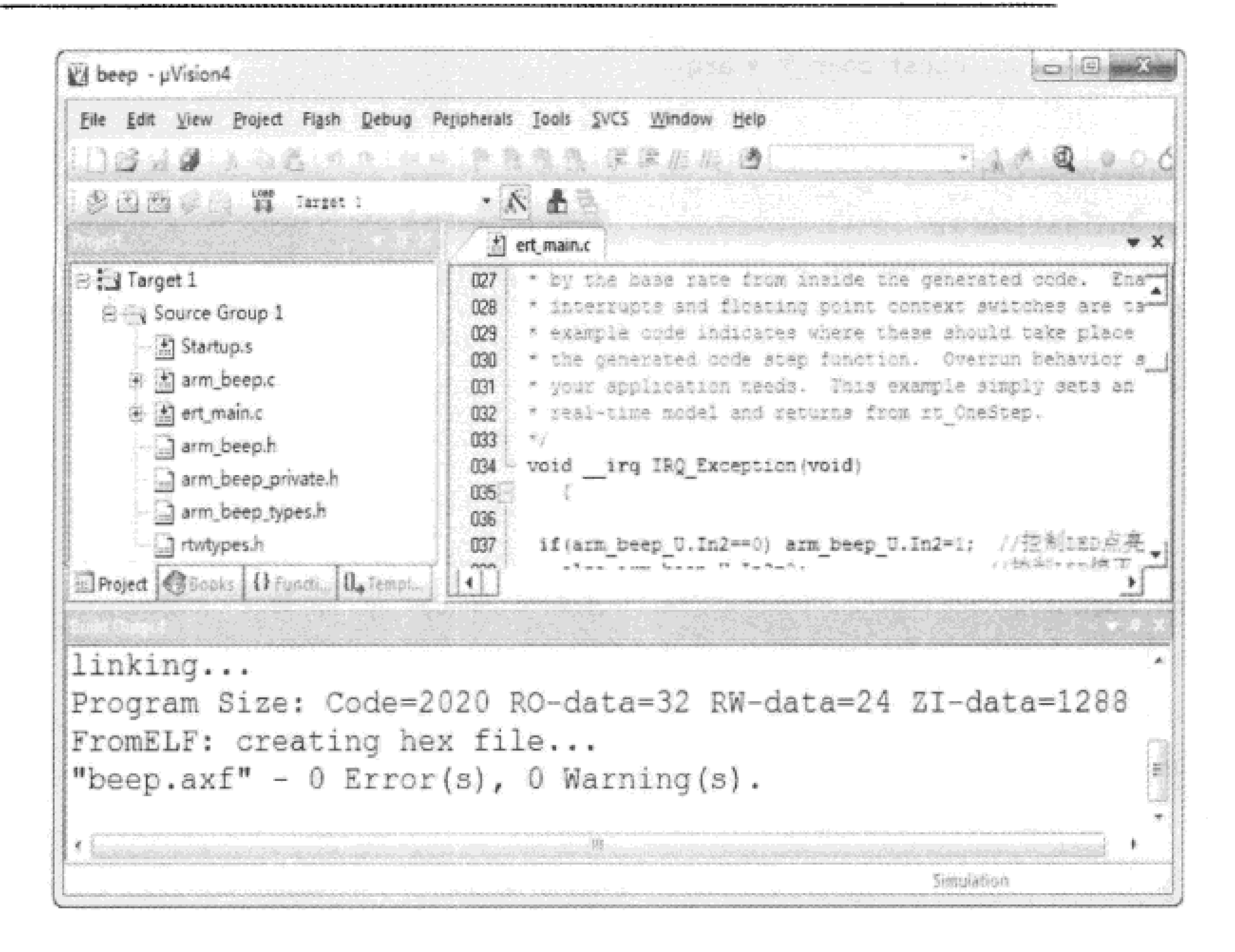


图 8.2.21 编译信息

8.2.5 虚拟硬件测试

参考第 8.1.2 小节的内容，建立蜂鸣器虚拟硬件模型，并加载先前生成的.hex 文件，单击“仿真”按钮。可以看到，当输入为 0 时，蜂鸣器停止工作，当输入为 1 时，芯片输出脉冲信号，驱动蜂鸣器发出声音，符合模型预期功能，如图 8.2.22 所示。

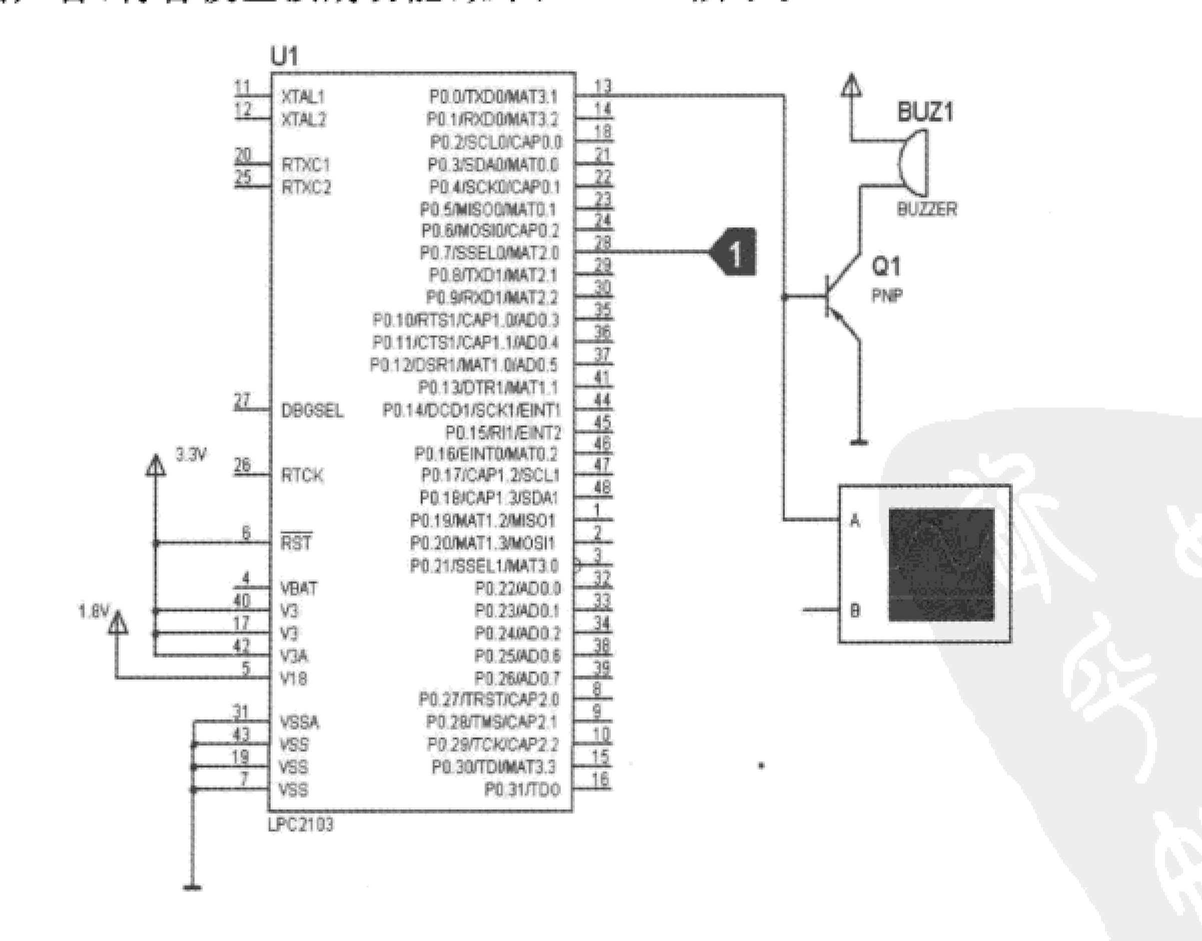
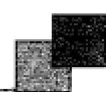


图 8.2.22 Proteus 原理图



输出信号波形如图 8.2.23 所示。

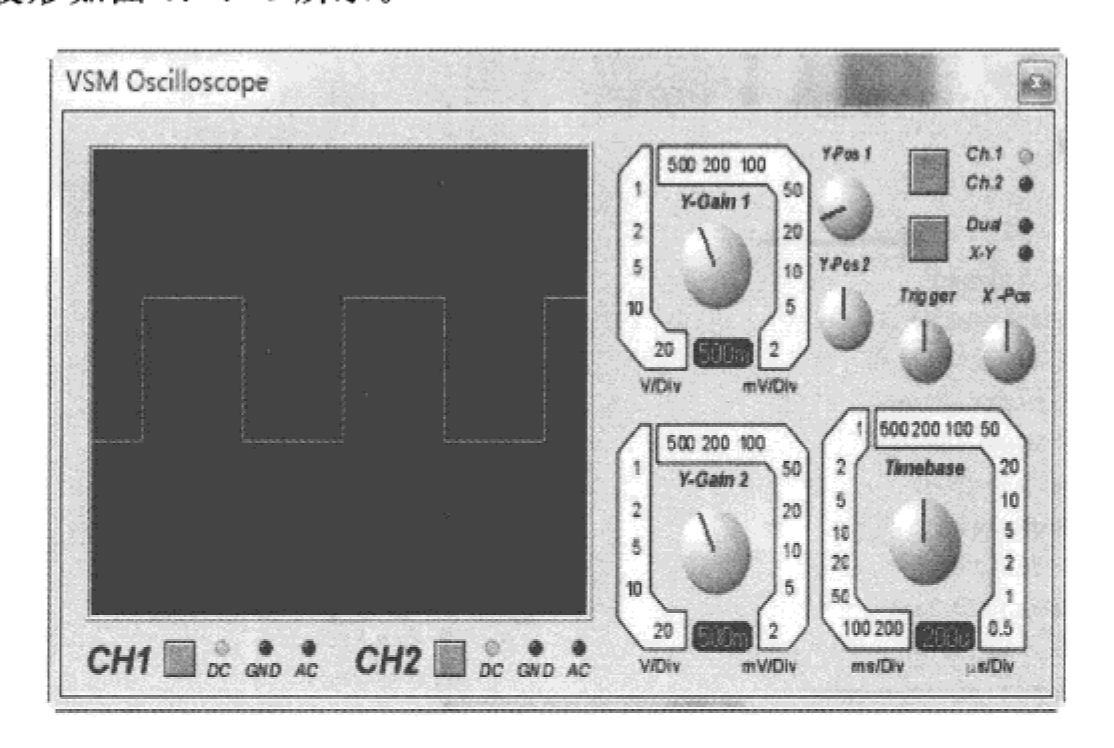


图 8.2.23 输出波形

8.3 交通灯控制

沿用 3.7.2 小节建立的交通灯模型，稍作修改即可在 ARM 处理器上实现其功能。将 Stateflow 模型重绘图 8.3.1(略去了 LED 灯与数码管显示部分)。

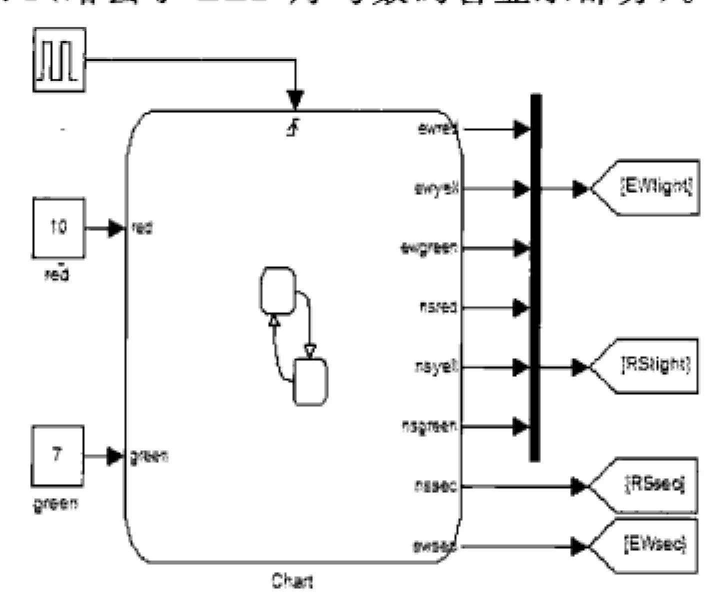


图 8.3.1 Simulink 功能验证模型

8.3.1 软件在环测试

在生成代码前，仍需要进行软件在环测试。

1. 数据类型转换

在模块库 Simulink→Ports & Subsystems 中找到输入模块与输出模块，替换图 8.3.1 中的脉冲生成器、常数、Goto 模块，其余的 LED 灯与数码管显示部分也应删去，并将模型另存。

单击模型窗口的按钮，打开模型浏览器如图 8.3.2 所示，依次设置 Simulink 模型、

Stateflow 状态图以及状态图中两个 Embedded MATLAB 函数的各变量数据类型。

Simulink 模型中的 In 模块的数据类型设置为 uint32，Out 模块的数据类型可设为自动继承，也可强制设置为 uint32，如图 8.3.3 所示。

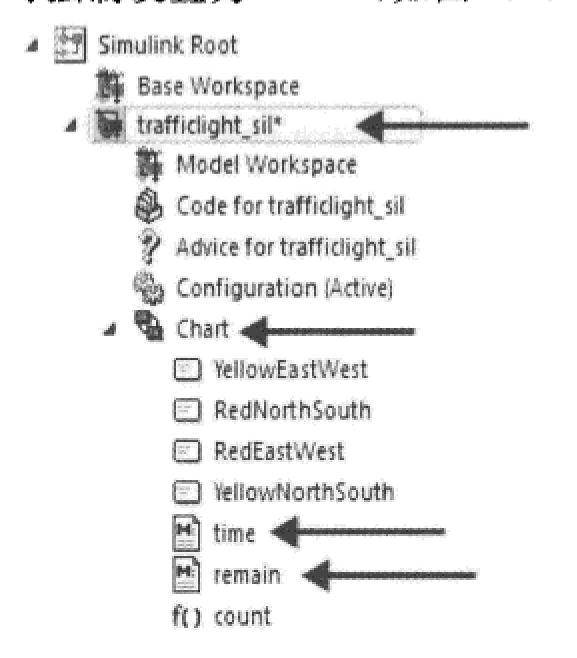
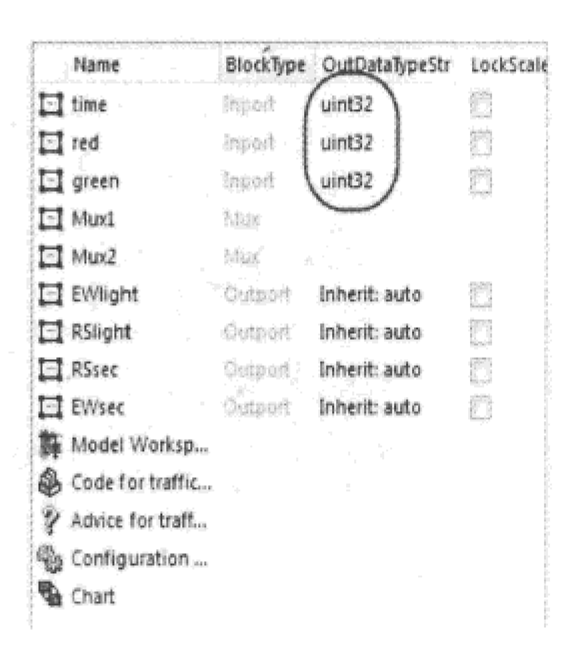


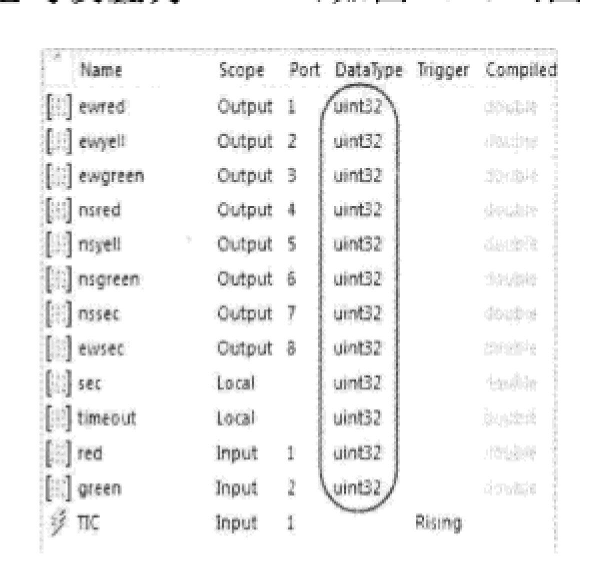
图 8.3.2　模型浏览器树状结构



Name	BlockType	OutDataTypeStr	LockScale
time	Inport	uint32	
red	Inport	uint32	
green	Inport	uint32	
Mux1	Mux		
Mux2	Mux		
EWlight	Outport	Inherit: auto	
RSlight	Outport	Inherit: auto	
RSsec	Outport	Inherit: auto	
EWsec	Outport	Inherit: auto	
Model Worksp...			
Code for traffic...			
Advice for traff...			
Configuration ...			
Chart			

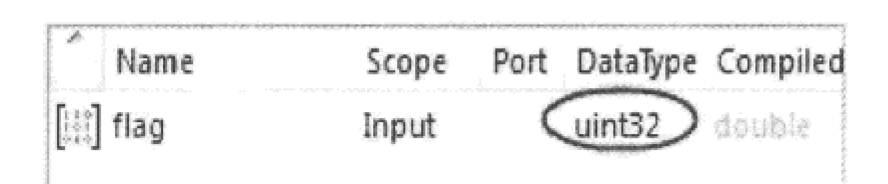
图 8.3.3　修改 Simulink 端口数据类型

Stateflow 状态图及所包含 Embedded MATLAB 函数 time(flag)、remain(x)的各变量数据类型均设置为 uint32，如图 8.3.4、图 8.3.5、图 8.3.6 所示。



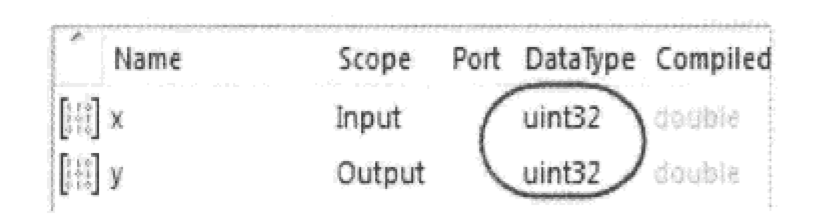
Name	Scope	Port	DataType	Trigger	Compiled
ewred	Output	1	uint32		double
ewyell	Output	2	uint32		double
ewgreen	Output	3	uint32		double
nsred	Output	4	uint32		double
nsyell	Output	5	uint32		double
nsgreen	Output	6	uint32		double
nssec	Output	7	uint32		double
ewsec	Output	8	uint32		double
sec	Local		uint32		double
timeout	Local		uint32		double
red	Input	1	uint32		double
green	Input	2	uint32		double
TIC	Input	1		Rising	

图 8.3.4　修改 Stateflow 参数数据类型(一)



Name	Scope	Port	DataType	Compiled
flag	Input		uint32	double

图 8.3.5　修改 Stateflow 参数数据类型(二)

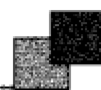


Name	Scope	Port	DataType	Compiled
x	Input		uint32	double
y	Output		uint32	double

图 8.3.6　修改 Stateflow 参数数据类型(三)

打开 Embedded MATLAB 函数 time(flag)，将其中所有常数的类型也转换为 uint32。

```
function time(flag)
timeout = uint32(0);                    % 转化为 uint32
switch flag
 case 1
    nssec = red;
    ewsec = green;
    sec = green * uint32(10);           % 转化为 uint32
 case 2
    ewsec = red-green;
    sec = (red-green) * uint32(10);     % 转化为 uint32
```



```
  case 3
     ewsec = red;
     nssec = green;
     sec = green * uint32(10);              % 转化为 uint32
  case 4
     nssec = red-green;
     sec = (red-green) * uint32(10);        % 转化为 uint32
end
```

修改后的模型如图 8.3.7 所示。

2. 模型参数设置

打开模型参数对话框，在 Real-Time Workshop 界面设置 TLC 文件为 ert.tlc，如图 8.3.8 所示。

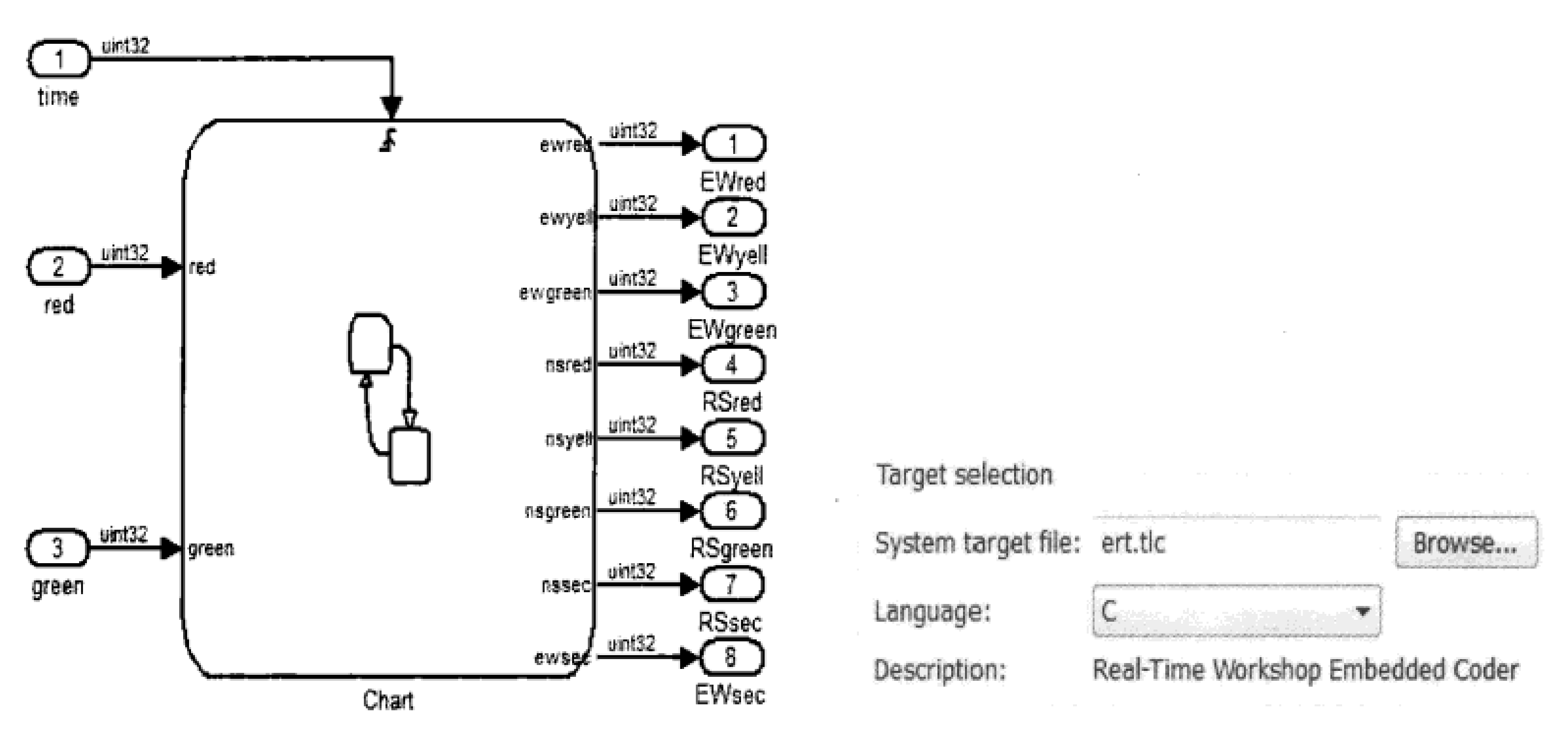


图 8.3.7　代码模型　　　　**图 8.3.8　设置 TLC**

Real-Time Workshop→Interface 界面，取消勾选不必要的复选框，如图 8.3.9 所示。

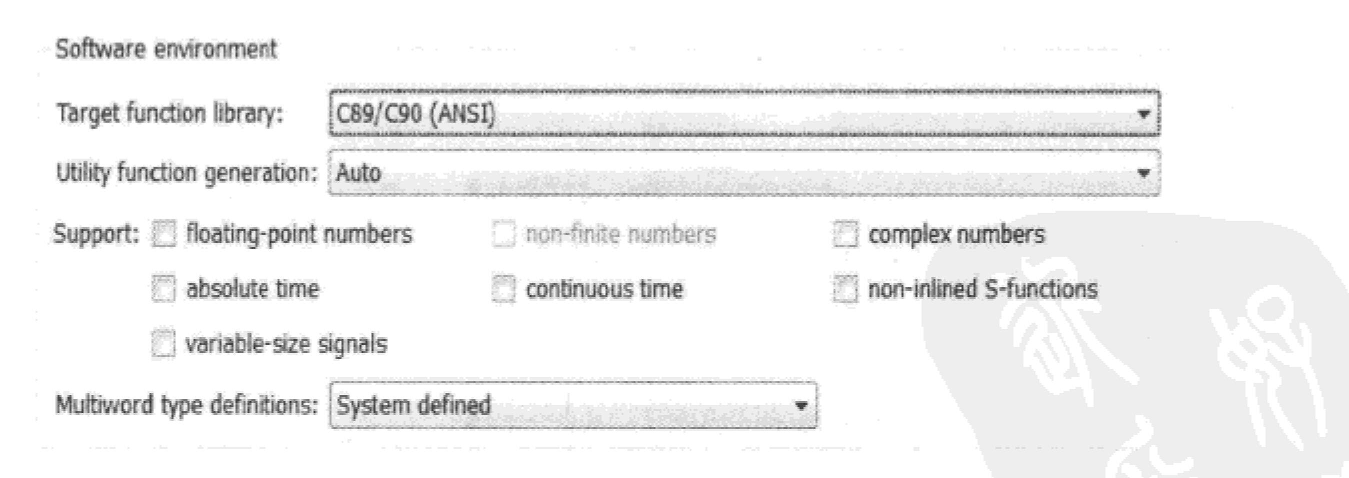


图 8.3.9　Interface 界面设置

Real-Time Workshop→Report 界面，勾选所有复选框，便于后期检查及跟踪，如图 8.3.10 所示。

3. 生成 SIL 模块

在 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表，选择

SIL 选项，如图 8.3.11 所示。

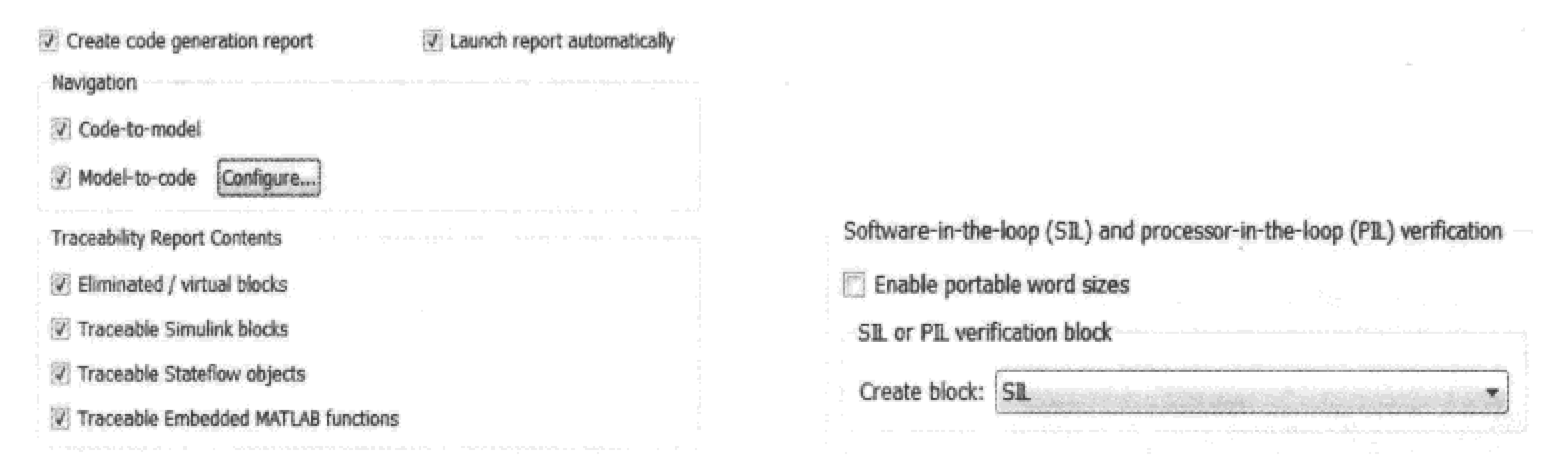
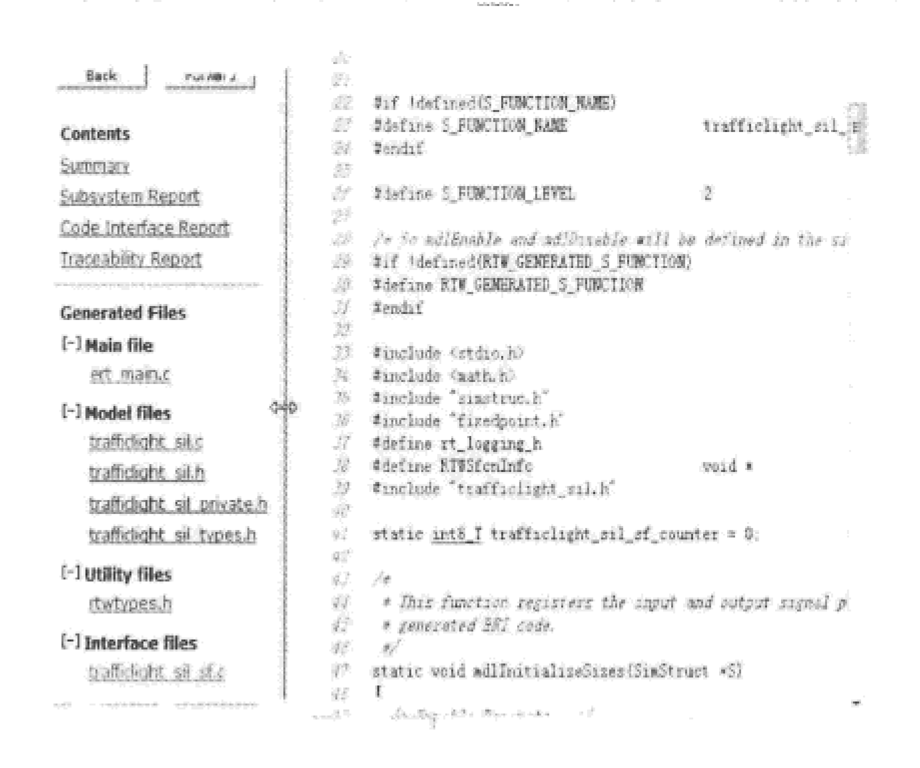


图 8.3.10　报告界面设置　　　　图 8.3.11　SIL 设置

单击模型工具栏的按钮，得到代码生成报告(图 8.3.12)与 SIL 模块(图 8.3.13)。



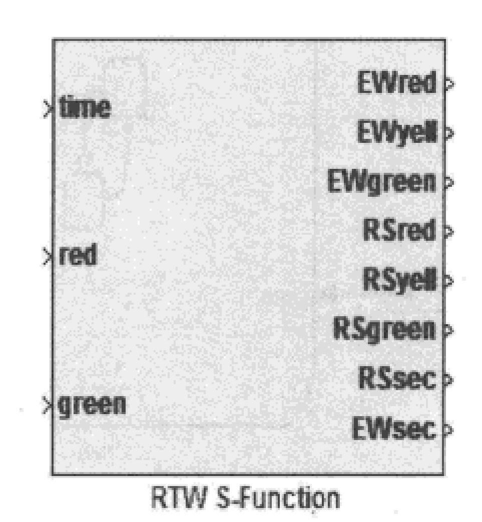


图 8.3.12　代码生成报告　　　　图 8.3.13　SIL 模块

按第 3.7.2 节中的 Simulink 功能验证模型，以 SIL 模块替换原有的 Stateflow 模块，重建图 8.3.14 所示的验证模型，并在各端口间加入必要的数据类型转换模块。该模型的运行结果与原模型是一致的。

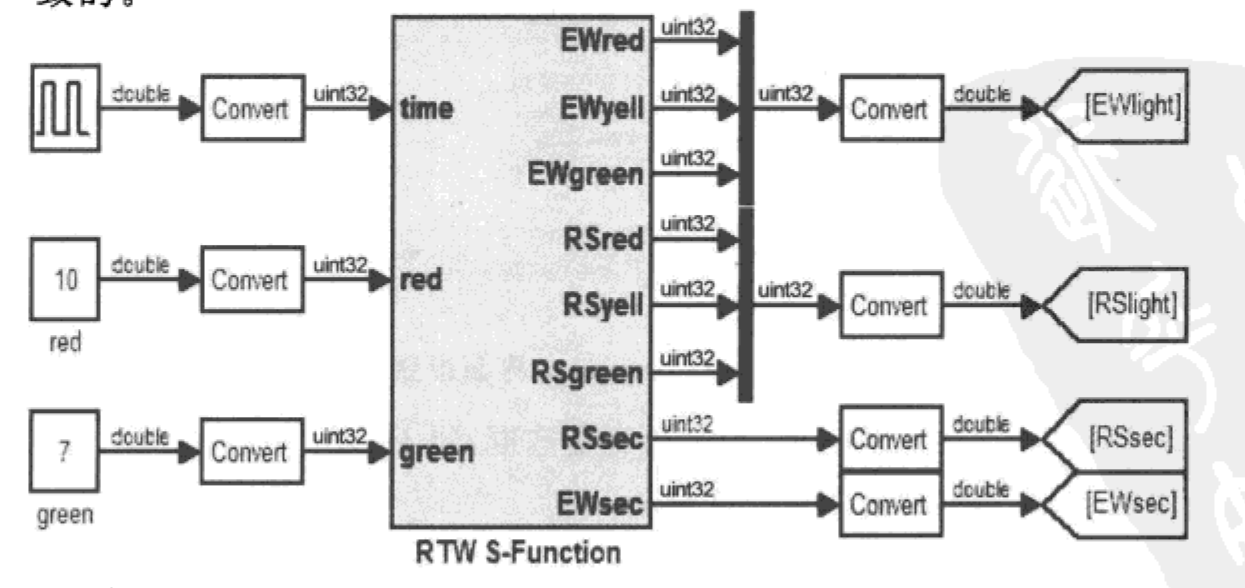
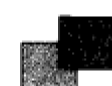


图 8.3.14　SIL 测试模型



8.3.2　自动代码生成及编译

1. 模型调整

为了适应 ARM 处理器的输出形式，编写 Embedded MATLAB 函数 combine，将 8 个输出端口合并为一个端口。函数内的各变量与常量的数据类型皆应设置为 uint32（图 8.3.15）。

Name	Scope	DataType	CompiledType	Port	Resolve Sign
nsred	Input	uint32	unknown	1	
nsyell	Input	uint32	unknown	2	
nsgre	Input	uint32	unknown	3	
ewred	Input	uint32	unknown	4	
ewyell	Input	uint32	unknown	5	
ewgre	Input	uint32	unknown	6	
nssec	Input	uint32	unknown	7	
ewsec	Input	uint32	unknown	8	
out	Output	uint32	unknown	1	☐

图 8.3.15　设置 Embedded MATLAB 函数变量的数据类型

代码如下：

```
function out =
combine(nsred,nsyell,nsgre,ewred,ewyell,ewgre,nssec,ewsec)
% #eml
led = ewgre * uint32(32) + ewyell * uint32(16) + ewred * uint32(8) + nsgre * uint32(4) +
nsyell * uint32(2) + nsred;
ns1 = rem(nssec,uint32(10));
ns2 = (nssec - ns1) / uint32(10);
ew1 = rem(ewsec,uint32(10));
ew2 = (ewsec - ew1) / uint32(10);
sec = ew2 * uint32(4096) + ew1 * uint32(256) + ns2 * uint32(16) + ns1;
out = led * uint32(65536) + sec;
```

另外为了简化，舍去原模型中由外部指定红灯与绿灯亮灯时长的功能，在 Embedded MATLAB 函数 time(flag)中直接指定，因此在函数中为 red 与 green 赋值。

```
function time(flag)
red = uint32(15);                  % 指定红灯时长
green = uint32(10);                % 指定绿灯时长
timeout = uint32(0);
switch flag
 case 1
    nssec = red;
    ewsec = green;
```

```
        sec = green * uint32(10);
    case 2
        ewsec = red-green;
        sec = (red-green) * uint32(10);
    case 3
        ewsec = red;
        nssec = green;
        sec = green * uint32(10);
    case 4
        nssec = red-green;
        sec = (red-green) * uint32(10);
end
```

调整后的代码生成模型如图 8.3.16 所示。

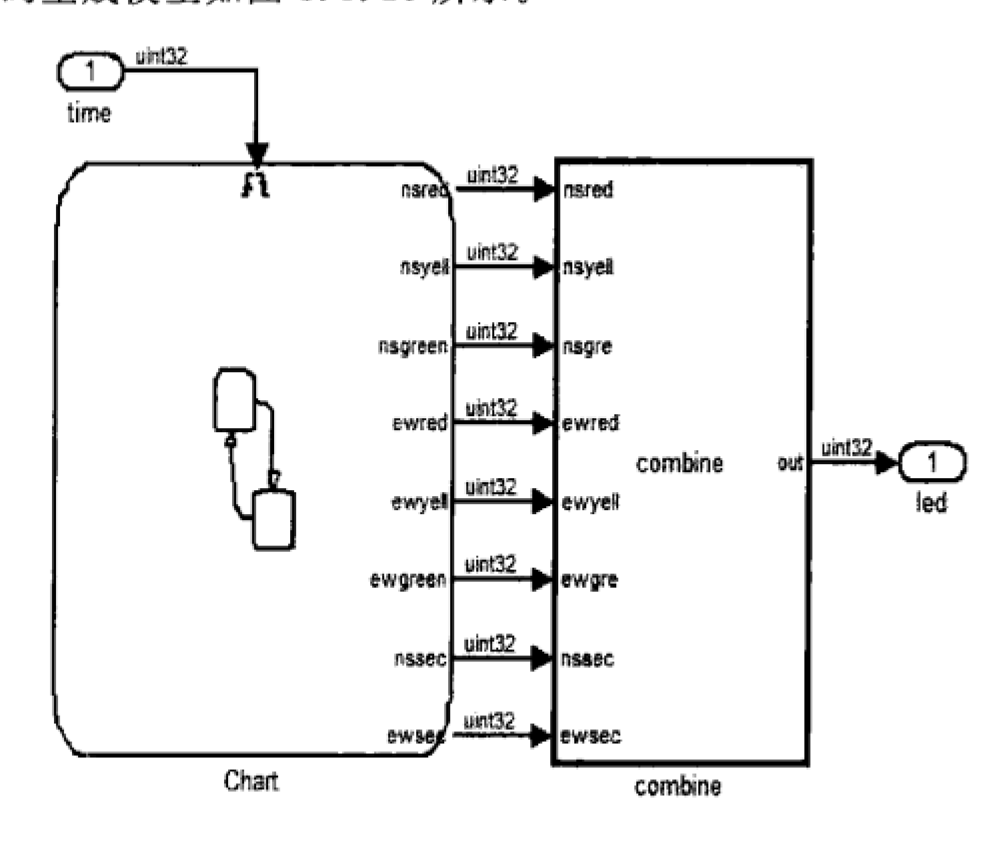


图 8.3.16　代码模型

2. 指定硬件

打开模型参数对话框，在 Hardware Implimentation 界面，设置器件类型为 ARM7，如图 8.3.17 所示。

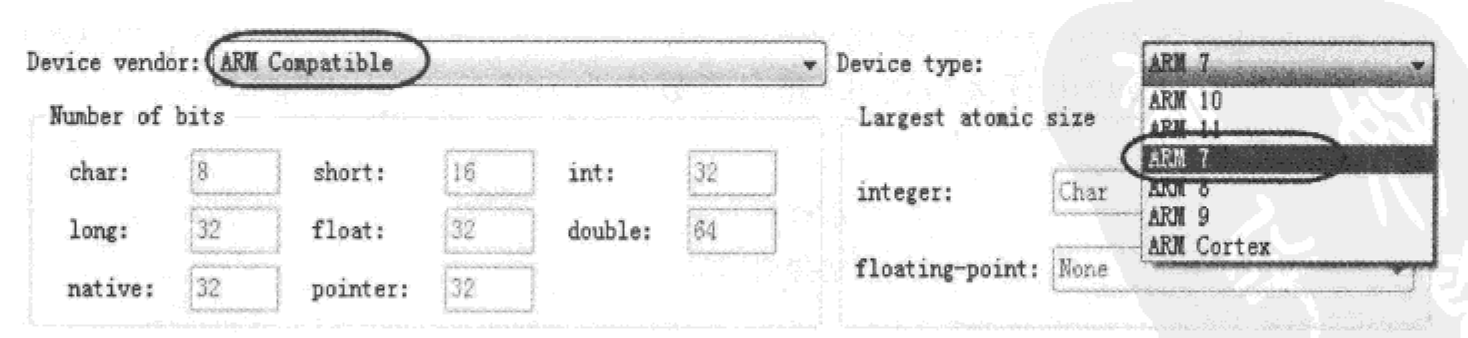
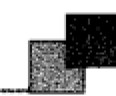


图 8.3.17　选择芯片

将 Real-Time Workshop→SIL and PIL Verification 界面的 Create block 下拉列表，选择 none 选项，如图 8.3.18 所示。



单击模型工具栏的按钮，生成代码，报告如图 8.3.19 所示。

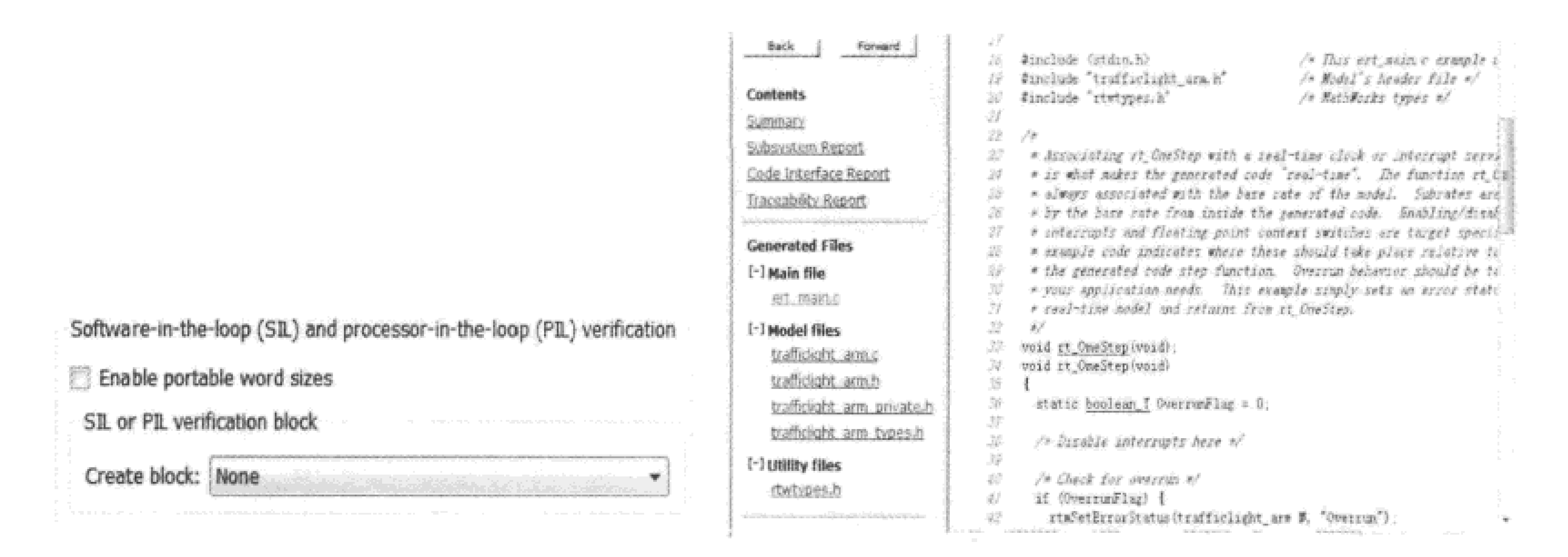


图 8.3.18　SIL 设置　　　　图 8.3.19　代码生成报告

3. 建立 Keil 工程

打开 Keil uVision4，建立基于 LPC2103 的工程，如图 8.3.20 所示。将图 8.3.19 所示的 6 个文件与 Keil 工程保存在同一目录下，这样在编译时，编译器会自动找到所需要的头文件，无需手工添加。

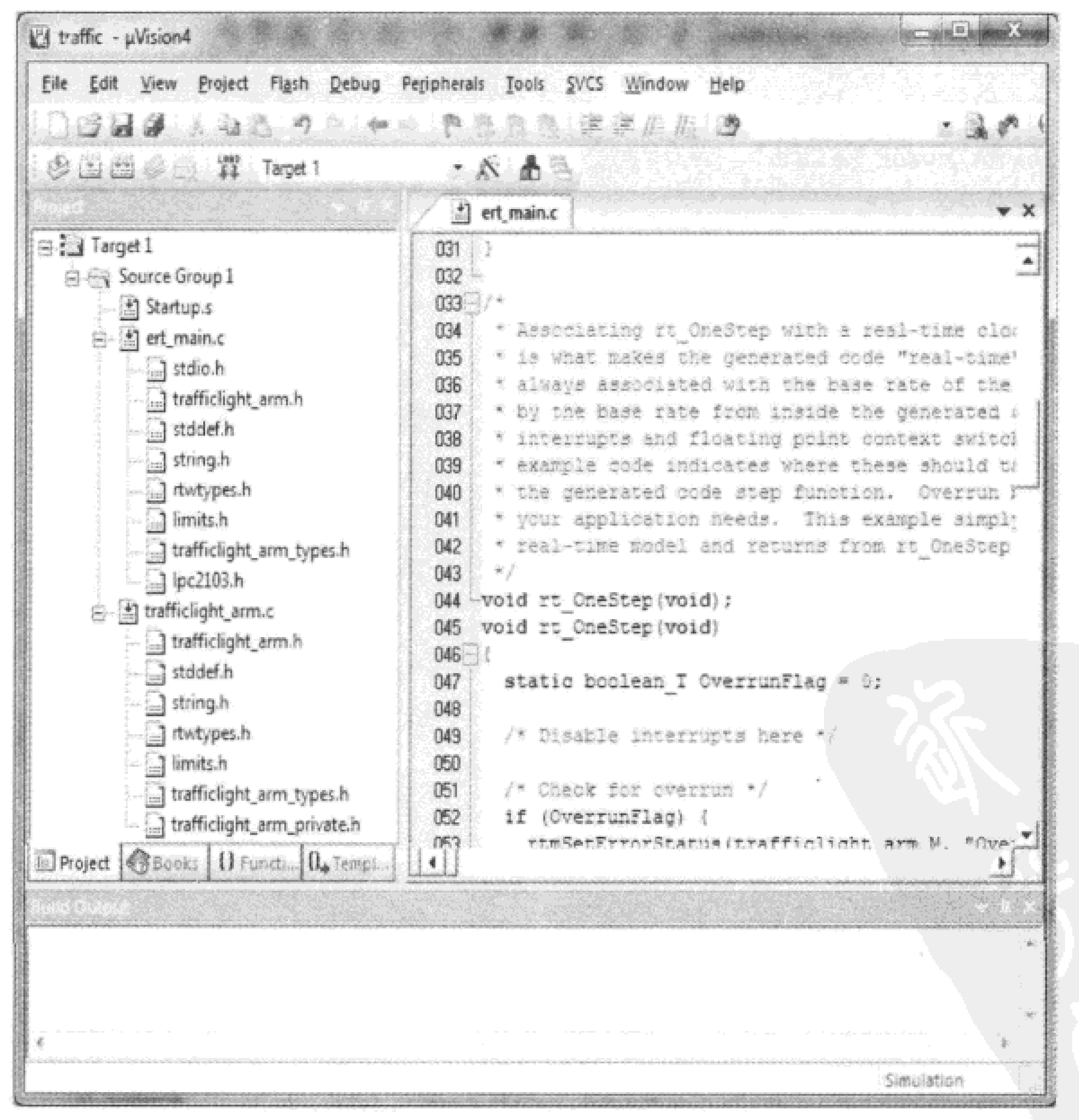


图 8.3.20　Keil 工程

单击 Keil 工具栏的按钮，或按 Alt+F7 组合键，在 Output 选项卡，勾选 Create HEX File 复选框，如图 8.3.21 所示。

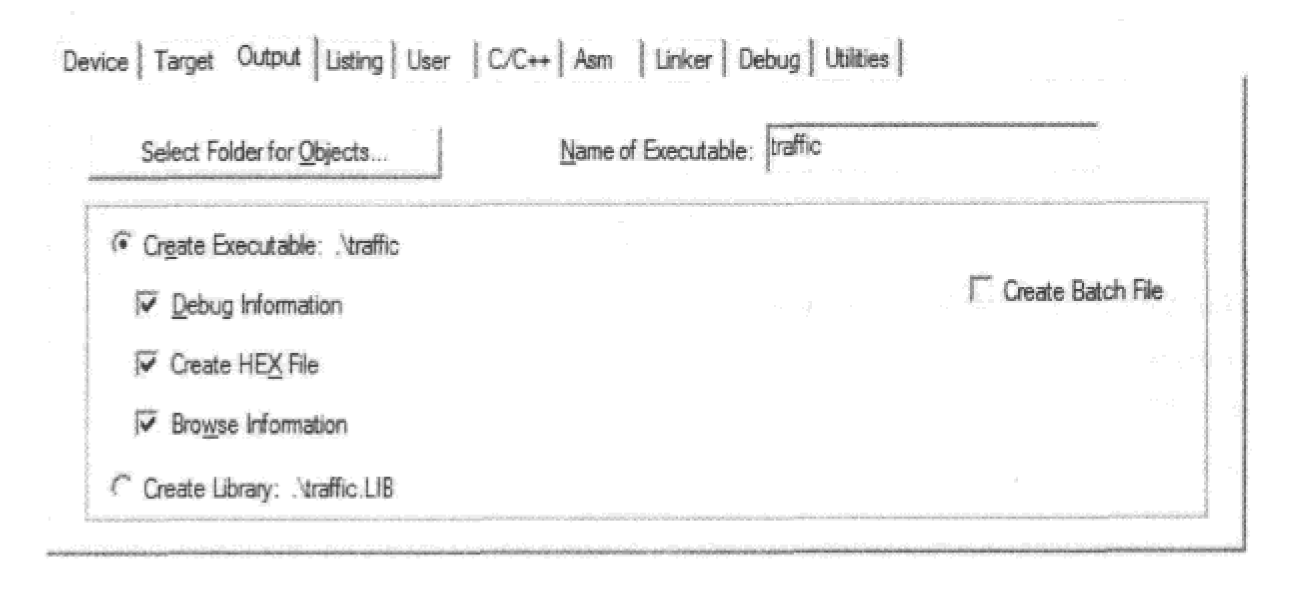


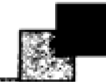
图 8.3.21 设置输出 HEX 文件

4. 代码修改

刚刚生成的代码实现了 Stateflow 算法，未包含脉冲发生器模块以及硬件接口，因此还需要使用定时器中断实现脉冲发生器模块，并将算法的输出与硬件端口相连接，修改的代码以斜体字表示如下：

```
...
#include <stdio.h>                              // 删除该头文件
#include "light_count_keil.h"
#include "rtwtypes.h"
#include <LPC2103.H>                            // 新增该头文件
void __irq t0(void)                             // 中断服务程序
{   T0TC = 0;                                   // 计时器清零
    T0IR = 0x01;                                // 清除中断标志
    IOCLR = 0x003FFFFF;                         // 输出清零
    trafficlight_arm_U.time = ~trafficlight_arm_U.time;
    VICVectAddr = 0x00;                         // 中断向量结束
}
```

```
void rt_OneStep(void);
void rt_OneStep(void)
{   ...
    // Step the model
    trafficlight_arm_step();
    // Get model outputs here
    IOSET = trafficlight_arm_Y.led;             //将输出与实际硬件端口连接
    ...
}
```



```
int_T main(int_T argc, const char_T *argv[]);
int_T main(int_T argc, const char_T *argv[])
{
    // Initialize model
    trafficlight_arm_initialize();
    PINSEL0  =  0x00000000;
    PINSEL1  =  0x00000000;
    IODIR  =  0x003FFFFF;                         // 设置 IO 为输出
    trafficlight_arm_U.time  =  0;// 触发信号初始为 0
    // 定时器初始化
    T0TCR  =  0x02;                               // 定时器 0 复位
    T0PR  =  99;                                  // 分频值
    T0MCR  =  0x03;                               // 匹配后复位 TC,并产生中断
    T0MR0  =  12500;                              // 定时匹配值与分频值,用户应根据实际硬件设置

    T0IR  =  0xFF;                                // 清除中断标志
    T0TCR  =  0x01;                               // 启动定时器 0
   // 中断向量初始化
    VICIntSelect  =  VICIntSelect & (~(1 <<4)); // 定时器 0 分配为 IRQ 中断
    VICVectCntl0  =  0x20 | 4;                    // 定时器 0 分配为向量 IRQ 通道 0
    VICVectAddr0  =  (uint32_T)t0;                // 分配中断服务程序地址
    VICIntEnable  =  1 <<4;                       // 定时器 0 中断使能
   //删除以下代码
   //printf(" Warning: The simulation will run forever. "
   //" Generated ERT main won't simulate model step behavior. "
   //" To change this behavior select the 'MAT-file
   //logging' option.\n");
   //fflush((NULL));
   while (rtmGetErrorStatus(trafficlight_arm_M) == (NULL))
   {   rt_OneStep();
   }
   ...
}
```

代码修改完成后单击工具栏按钮,编译工程,如图 8.3.22 所示,窗口下部的信息显示已成功生成.hex 文件。

8.3.3 虚拟硬件测试

仿造 3.7.2 小节的 Simulink 模型,根据本章第 5.1 节的介绍,建立 proteus 交通灯模型,并加载先前生成的 HEX 文件。单击“仿真”按钮,模型即按预设的时间亮灯,实现了 Simulink 模型所设计的功能,如图 8.3.23～图 8.3.26 所示。

traffic - µVision4

File Edit View Project Flash Debug Peripherals Tools SVCS Window Help

Target 1

- Target 1
  - Source Group 1
    - Startup.s
    - ert_main.c
      - trafficlight_arm.h
      - stddef.h
      - string.h
      - rtwtypes.h
      - limits.h
      - trafficlight_arm_types.h
      - lpc2103.h
    - trafficlight_arm.c
      - trafficlight_arm.h
      - stddef.h
      - string.h
      - rtwtypes.h
      - limits.h
      - trafficlight_arm_types.h
      - trafficlight_arm_private.h

Project | Books | {} Functi... | 0→ Templ...

ert_main.c

```
001 /*
002  * File: ert_main.c
003  *
004  * Real-Time Workshop code generated for Simul:
005  *
006  * Model version                        : 1.26:
007  * Real-Time Workshop file version      : 7.5
008  * Real-Time Workshop file generated on : Tue :
009  * TLC version                          : 7.5
010  * C/C++ source code generated on       : Tue :
011  *
012  * Target selection: ert.tlc
013  * Embedded hardware selection: ARM Compatible-
014  * Code generation objectives: Unspecified
015  * Validation result: Not run
016  */
017
018 //#include <stdio.h>                      /* Th:
019 #include "trafficlight_arm.h"             /* Mode:
020 #include "rtwtypes.h"                     /* Math:
021 #include <LPC2103.H>
022
```

```
Build target 'Target 1'
assembling Startup.s...
compiling ert_main.c...
compiling trafficlight_arm.c...
linking...
Program Size: Code=4144 RO-data=32 RW-data=36 ZI-data=1292
FromELF: creating hex file...
"traffic.axf" - 0 Error(s), 0 Warning(s).
```

Simulation

图 8.3.22　编译 Keil 工程

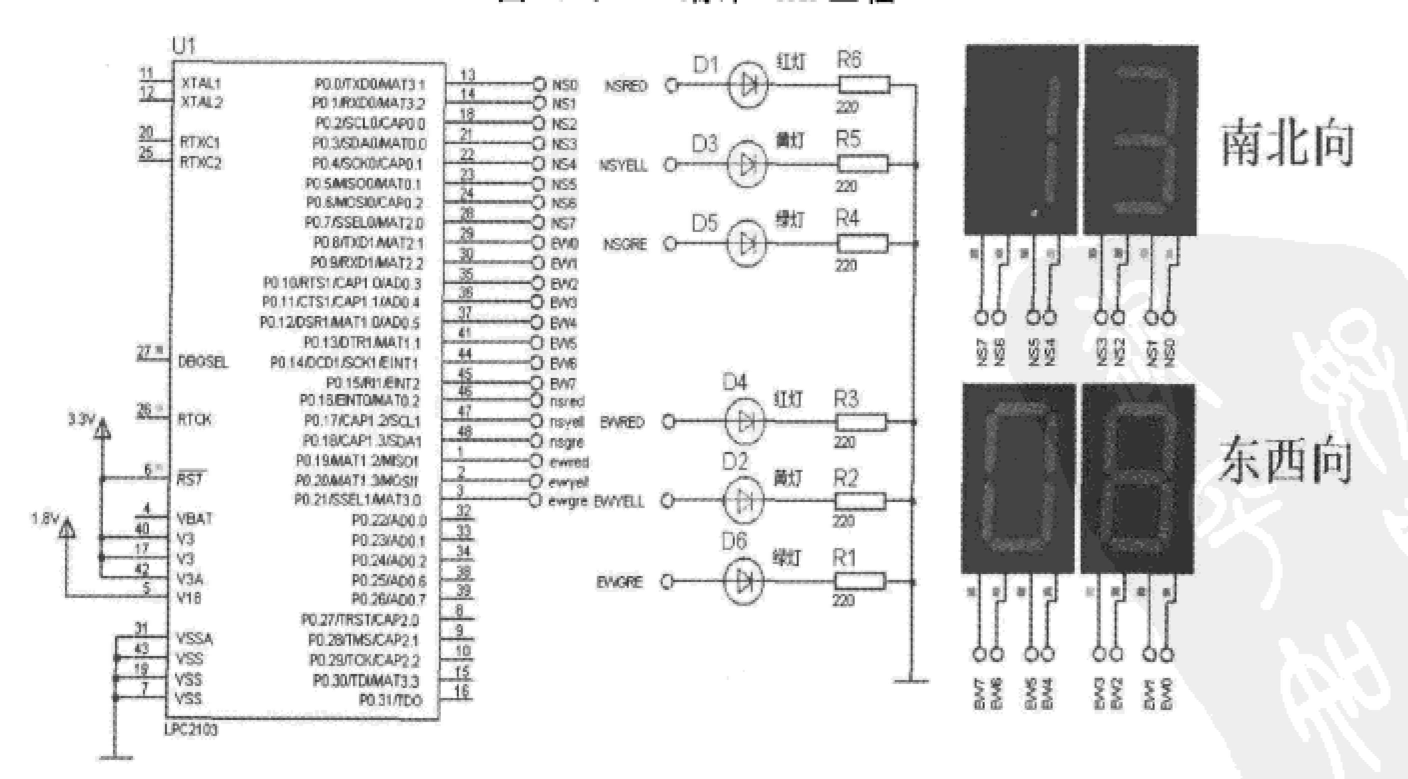
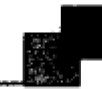


图 8.3.23　南北向禁行



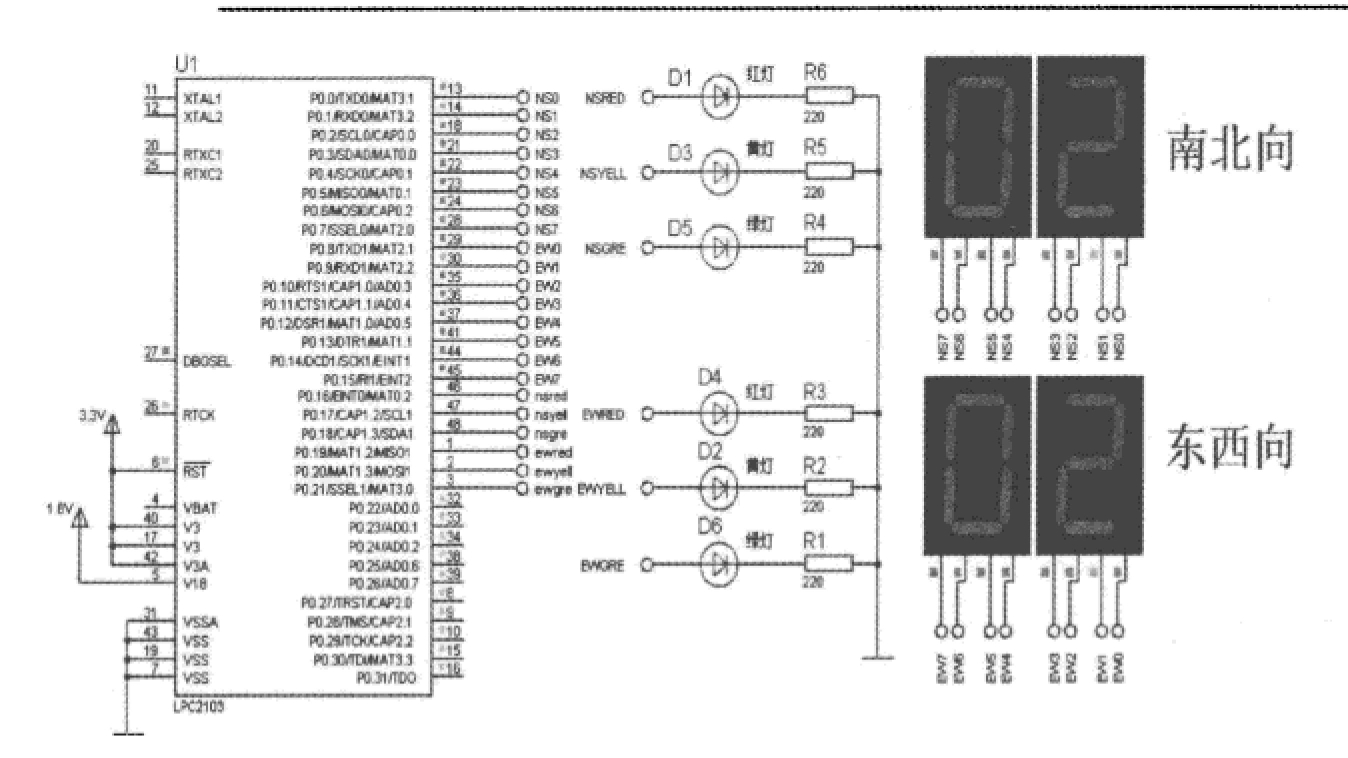


图 8.3.24　东西向缓行

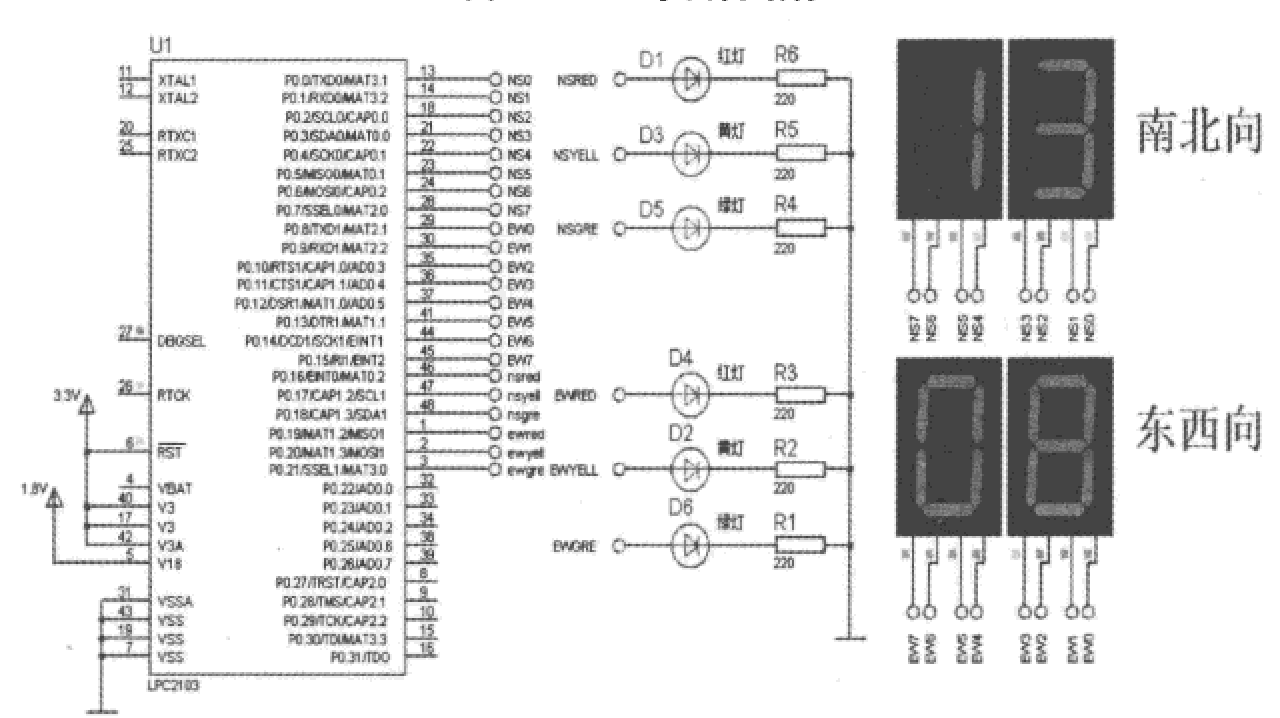


图 8.3.25　东西向禁行

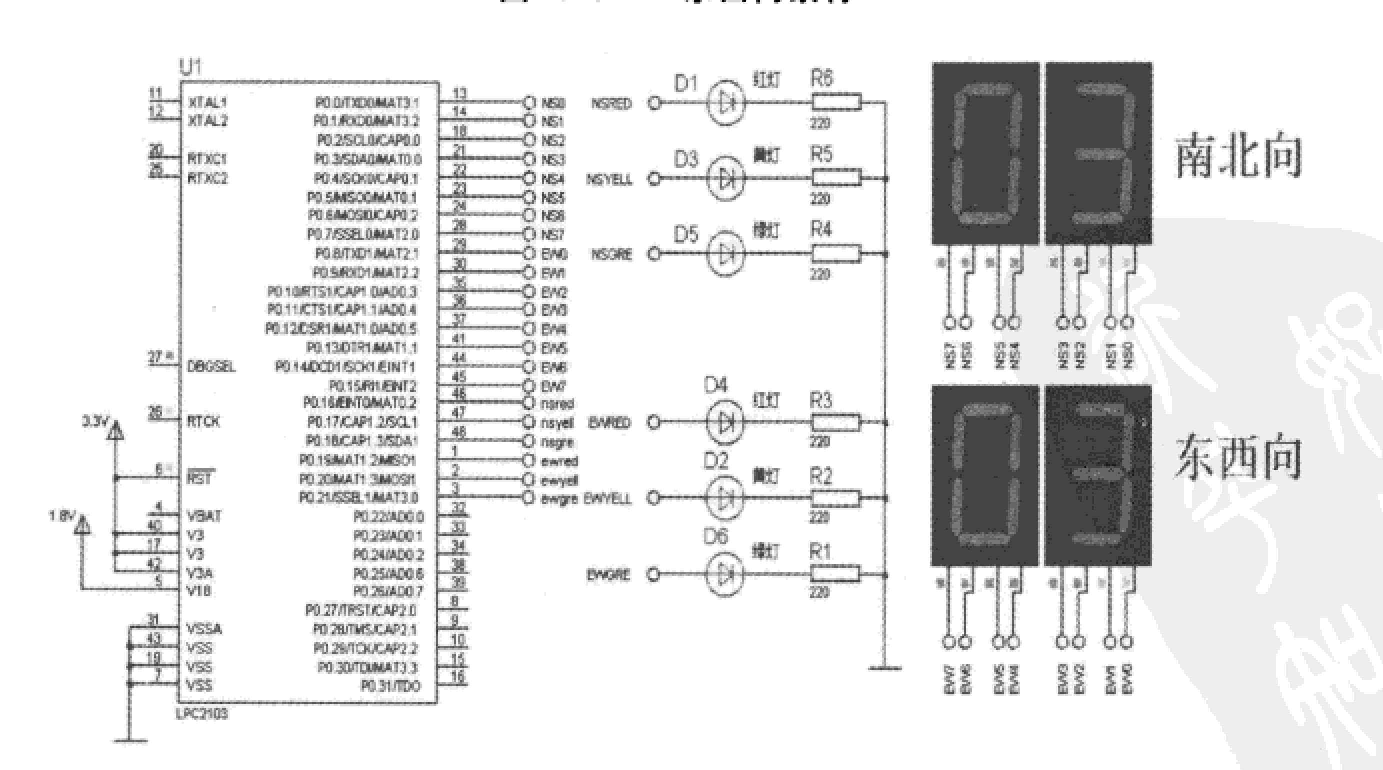


图 8.3.26　南北向缓行

8.4 步进电动机控制

8.4.1 步进电动机原理简介

步进电动机是机电一体化产品的典型代表,在精密仪器制造,数控机床等工业控制系统中有着广泛应用。

步进电动机不同于普通直流电动机,它不是连续转动的,而是每当驱动器接收到一个脉冲信号,它就驱动步进电动机按设定的方向转动一个固定的角度,故称为步进电动机。用户可以通过控制脉冲个数来控制角位移量,从而实现精确定位;通过控制脉冲频率来控制电动机转动的速度和加速度,从而实现调速。

步进电动机按相数可分为单相、双相及多相。Proteus 中的步进电动机是以四相步进电动机为模型建立的,下面针对该电动机作原理说明。

该电动机的转子为永磁体,定子为电磁线圈。当定子绕阻中的某一相有电流流过时,绕阻会产生磁场,该磁场会使转子旋转一个角度,使转子的磁场方向与其保持一致。然后,下一个绕阻通电,绕阻磁场随之转过一个角度,转子再次旋转至绕阻磁场方向。这样,每输入一个脉冲,电动机就会转动一个角度,连续不断的脉冲输入就可以驱动电动机持续旋转了。

在实际应用中,可以采取多种不同的励磁方式。例如,首先给 A 绕阻通电,转子转动到 A 绕阻的磁场方向;然后同时给 A,B 两个绕阻通电,则同时在 A,B 两个方向上产生磁场,转子会转动到 A,B 之间的平衡位置;接下来个 A 绕阻断电,B 继续保持导通状态,这样转子转动到 B 绕阻的磁场方向;按照类似的顺序,导通 B,C—C—CD—D—DA—A,电动机同样可以持续旋转,这种励磁方式可以减小电动机的步进角,有效提高控制精度。如图 8.4.1~图 8.4.3 所示。

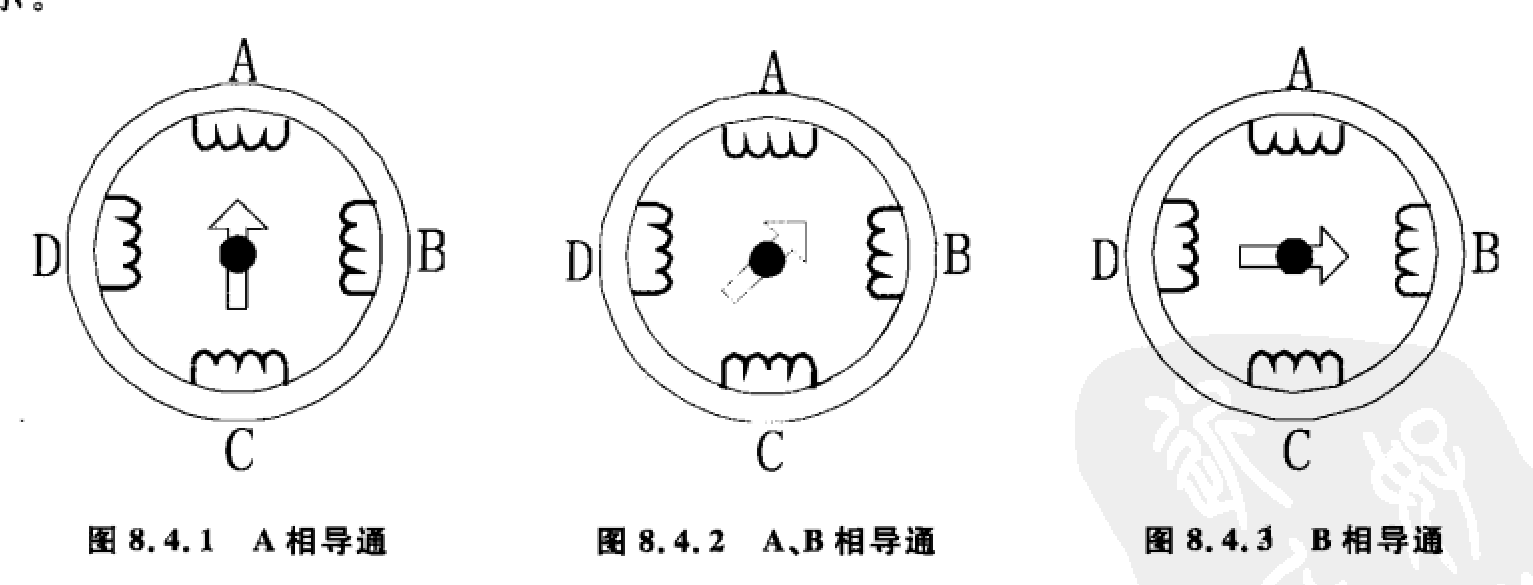
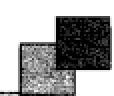


图 8.4.1 A 相导通　　图 8.4.2 A、B 相导通　　图 8.4.3 B 相导通

8.4.2 步进电动机控制模型

本例用四相八拍的方式驱动四相步进电动机。设 4 个绕阻分别为 A、B、C、D,则其通电顺序为 A-AB-B-BC-C-CD-D-DA。根据第 3 章的介绍,容易建立驱动步进电动机的状态图(图 8.4.4)。



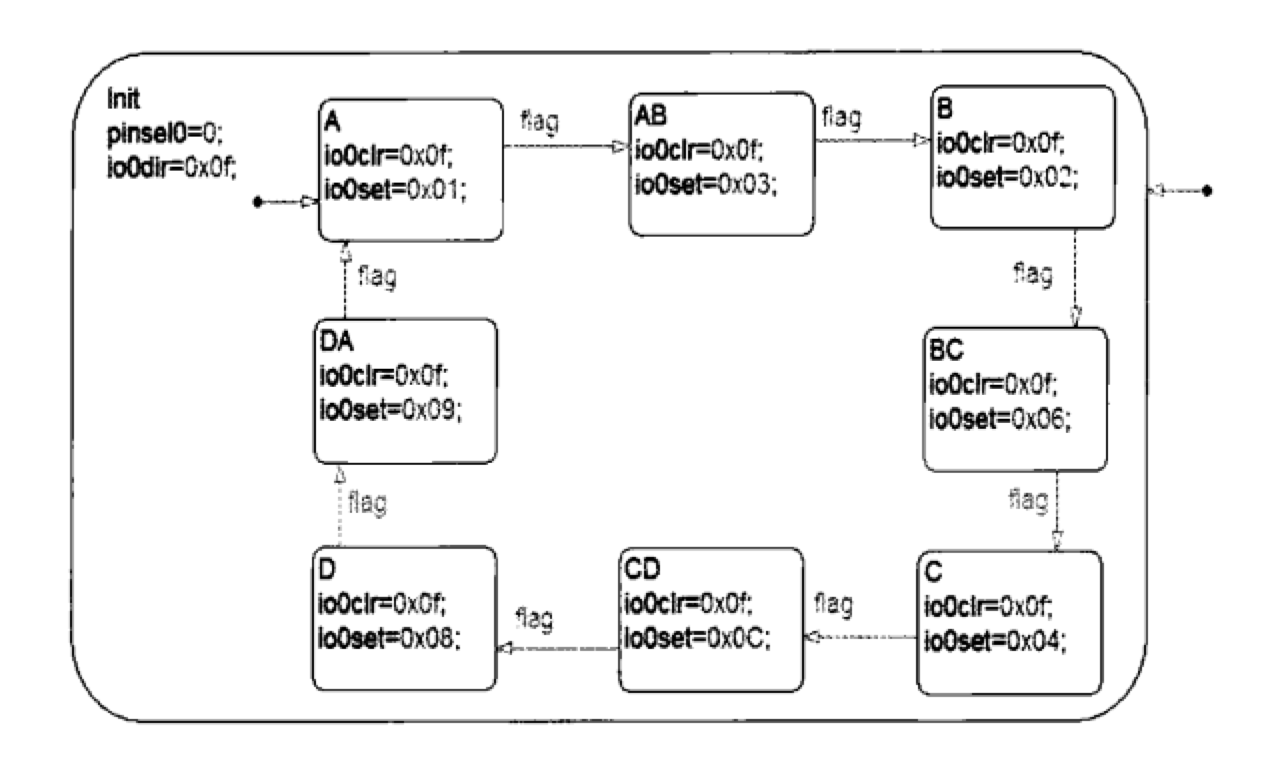


图 8.4.4　步进电动机驱动模型

其中数据 io0set、io0clr 以十六进制形式表示通电的绕阻；pinsel0、io0dir 为状态设置；事件 flag 表示绕阻通、断电转换标志，如图 8.4.5 所示。

Name	Scope	Port	Resolve Signal	DataType
flag	Input	1		
pinsel0	Output	1	☐	double
io0dir	Output	2	☐	double
io0set	Output	3	☐	double
io0clr	Output	4	☐	double

图 8.4.5　步进电动机驱动模型数据列表

8.4.3　步进电动机的功能验证模型

完成步进电动机驱动模型之后，在 Simulink 模块库中找到图 8.4.6、图 8.4.7 所示模块，并按图 8.4.8 所示连接。

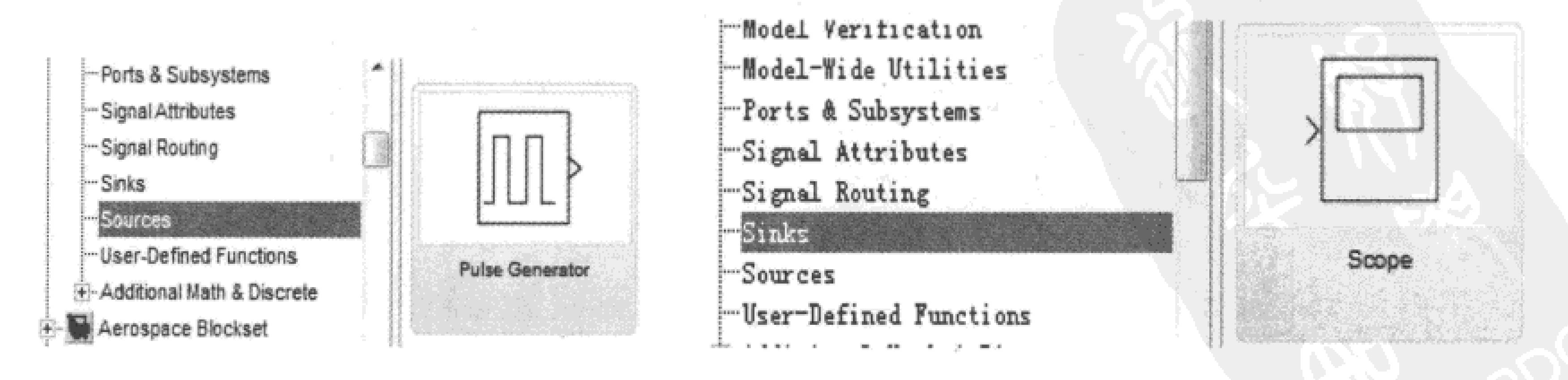


图 8.4.6　脉冲发生器模块　　**图 8.4.7　示波器模块**

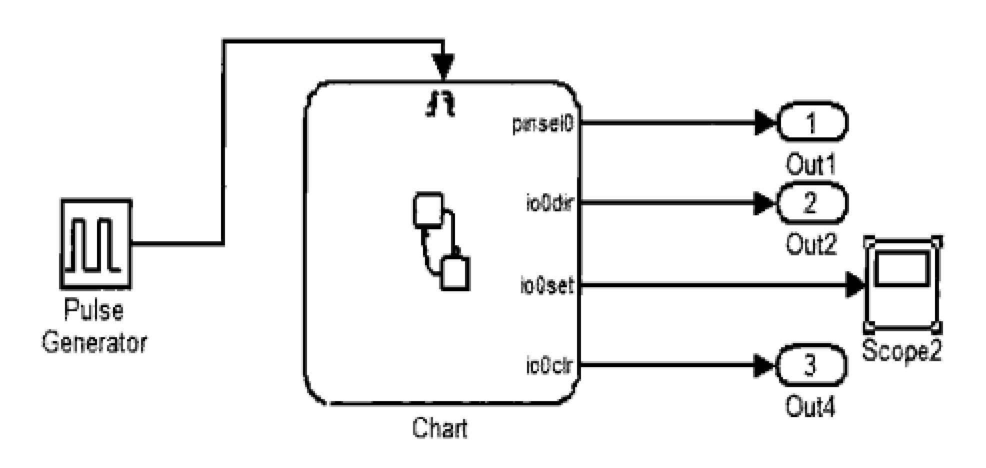


图 8.4.8 功能验证模型

选择模型主窗口的菜单项 Simulation→Configuration Parameters...，打开模型参数对话框，在 Solver options 面板中，设置求解器为定步长离散求解器，步长为 0.01，如图 8.4.9 所示。

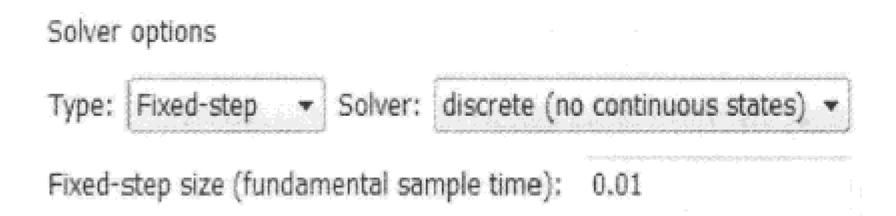


图 8.4.9 求解器设置

Pulse Generator 模块产生脉冲，使 flag 不断变化。运行模型，可以看到 io0set 端口的输出信号在一个周期内有 8 种状态，如图 8.4.10 所示。

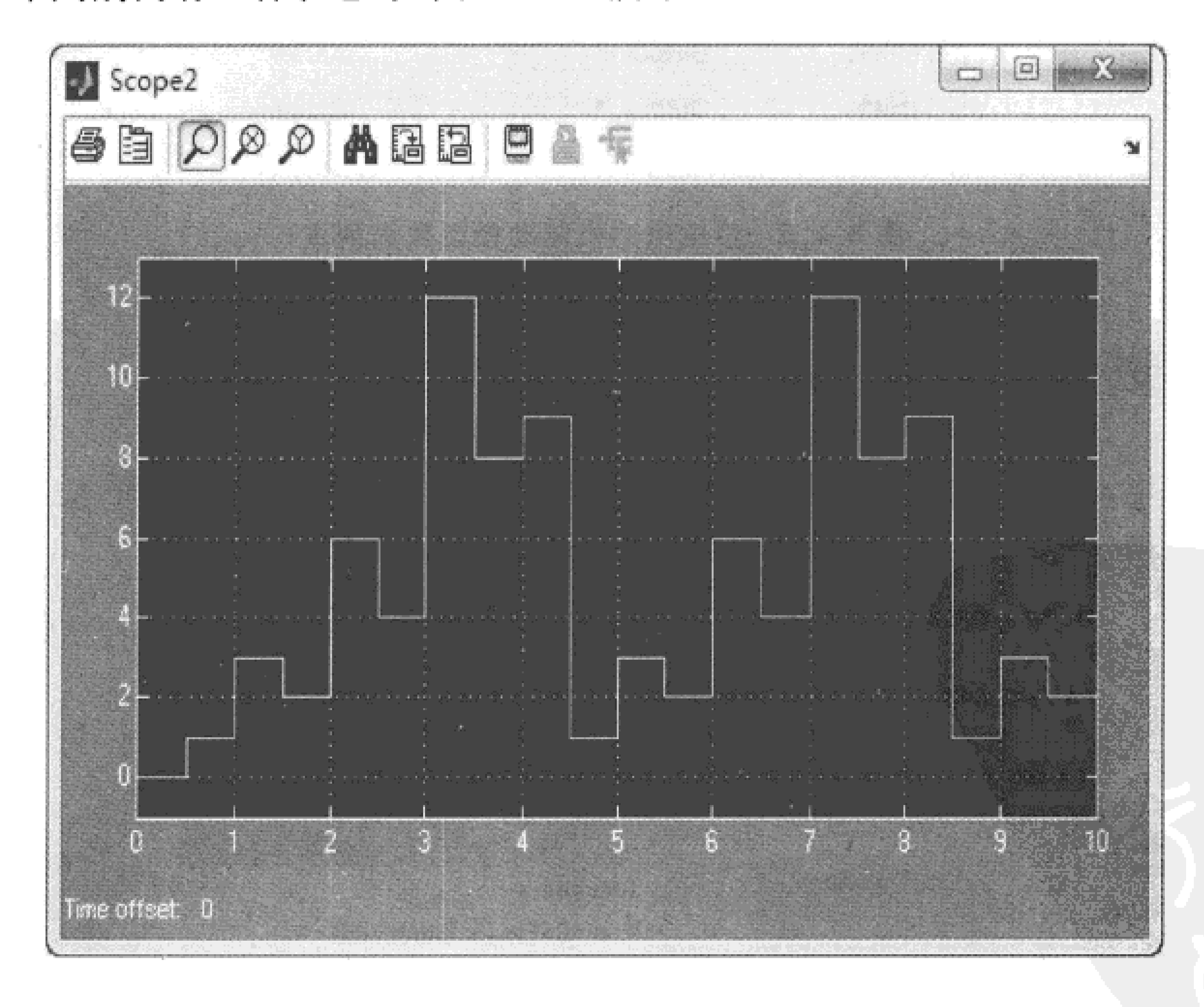
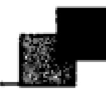


图 8.4.10 仿真结果



Io0set 值为 1 时对应于二进制数 0001，即 A 绕阻导通；值为 3 时对应于二进制数 0011，即 A，B 绕阻导通；值为 2 时对应于二进制数 0010，即 B 绕阻导通；值为 6 时对应于二进制数 0110，即 B，C 绕阻导通；值为 4 时对应于二进制数 0100，即 C 绕阻导通；值为 12 时对应于二进制数 1100，即 C，D 绕阻导通；值为 8 时对应于二进制数 1000，即 D 绕阻导通；值为 1 时对应于二进制数 1001，即 D、A 绕阻导通。

8.4.4　软件在环测试

在模块库 Simulink / Ports &Subsystems 中找到输入模块，替换图 8.4.8 中的脉冲信号源与常量输出，并将这些端口的数据类型修改为 uint32，如图 8.4.11 所示。

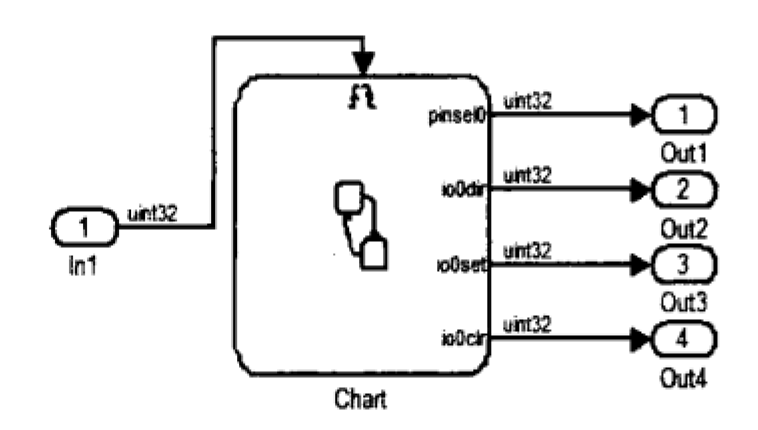


图 8.4.11　代码模型

另外还需要确保 Stateflow 模型中数据的类型也应为 uint32，如图 8.4.12、图 8.4.13 所示。

Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockScale	DataType
Model Worksp...						
Code for arm_...						
Advice for arm...						
Configuration ...						
Chart						
Out1	Outport	uint32	[]	[]		
Out2	Outport	uint32	[]	[]		
Out3	Outport	uint32	[]	[]		
Out4	Outport	uint32	[]	[]		
In1	Inport	uint32	[]	[]		

图 8.4.12　修改模型端口数据类型

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	Compilе
Init								
flag	Input	1						
io0dir	Out...	2		uint32			uint32	1
io0set	Out...	3		uint32			uint32	1
io0clr	Out...	4		uint32			uint32	1
pinsel0	Out...	1		uint32			uint32	1

图 8.4.13　修改模型内部数据类型

在 Report 界面中，勾选所有复选框，便于后期检查及跟踪，如图 8.4.14 所示。

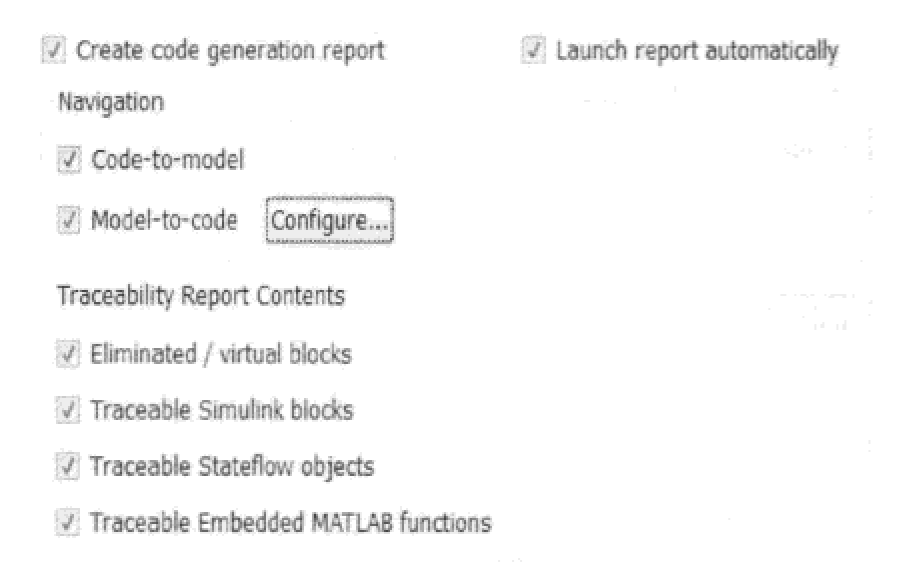


图 8.4.14　报告设置界面

在 Real-Time Workshop 界面中，设置 TLC 文件为 ert.tlc，如图 8.4.15 所示。

打开模型的参数设置对话框，在 Real-Time Workshop→SIL and PIL Verification 界面，勾选 Create SIL block 复选框，如图 8.4.16 所示。

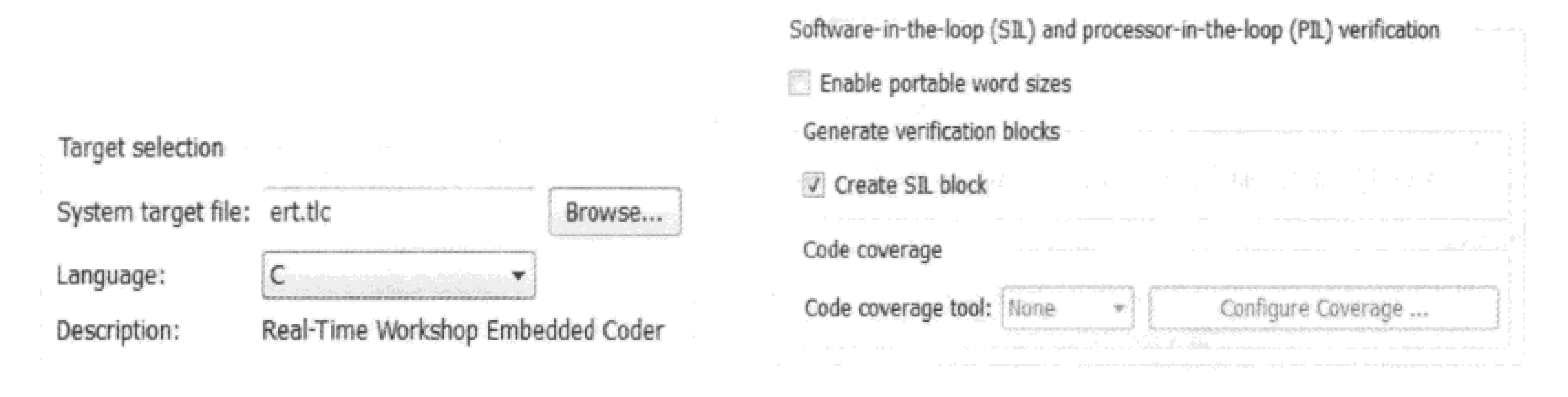


图 8.4.15　设置 TIC　　　　图 8.4.16　SIL 设置

单击模型工具栏的按钮，生成 SIL 模块，如图 8.4.17 所示。

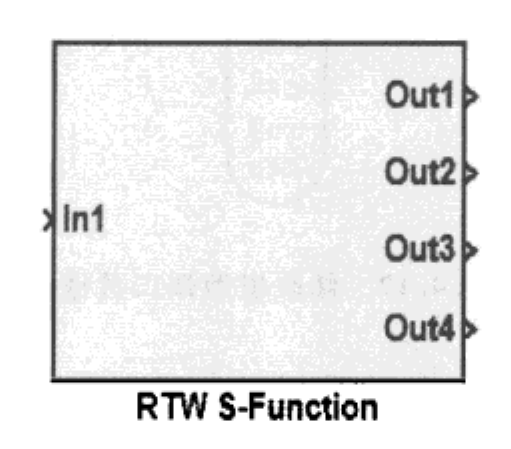
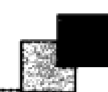


图 8.4.17　SIL 模块

如图 8.4.8 所示，以 SIL 模块替换 Stateflow 模块，重建图 8.4.18 所示的验证模型，由于 SIL 模块是根据定点模型建立的，因此各端口间加入了数据类型转换模块。

该模型的运行结果与图 8.4.10 所示的结果是一致的，如图 8.4.19 所示。



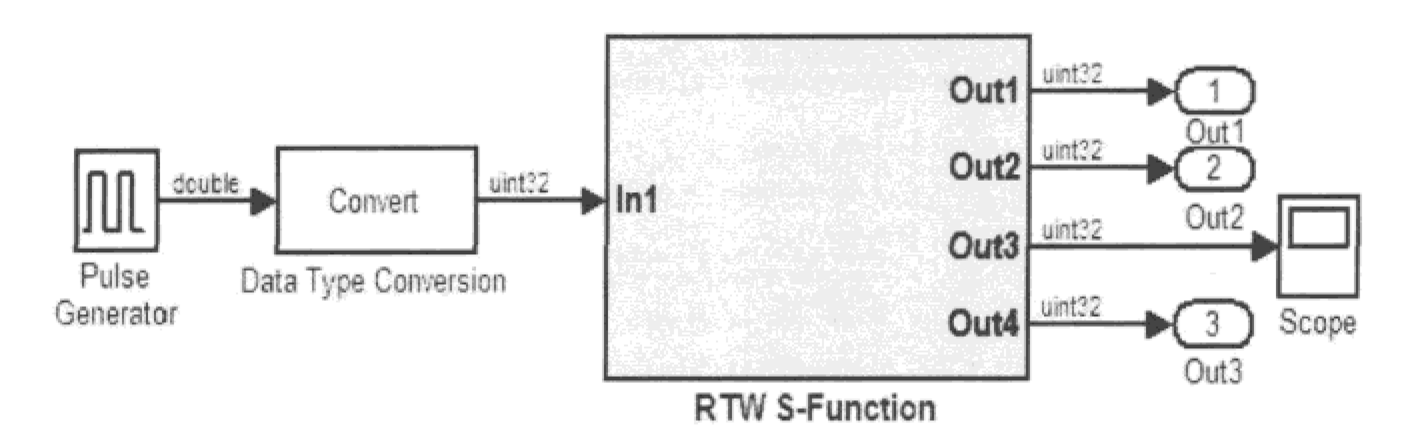


图 8.4.18　SIL 测试模型

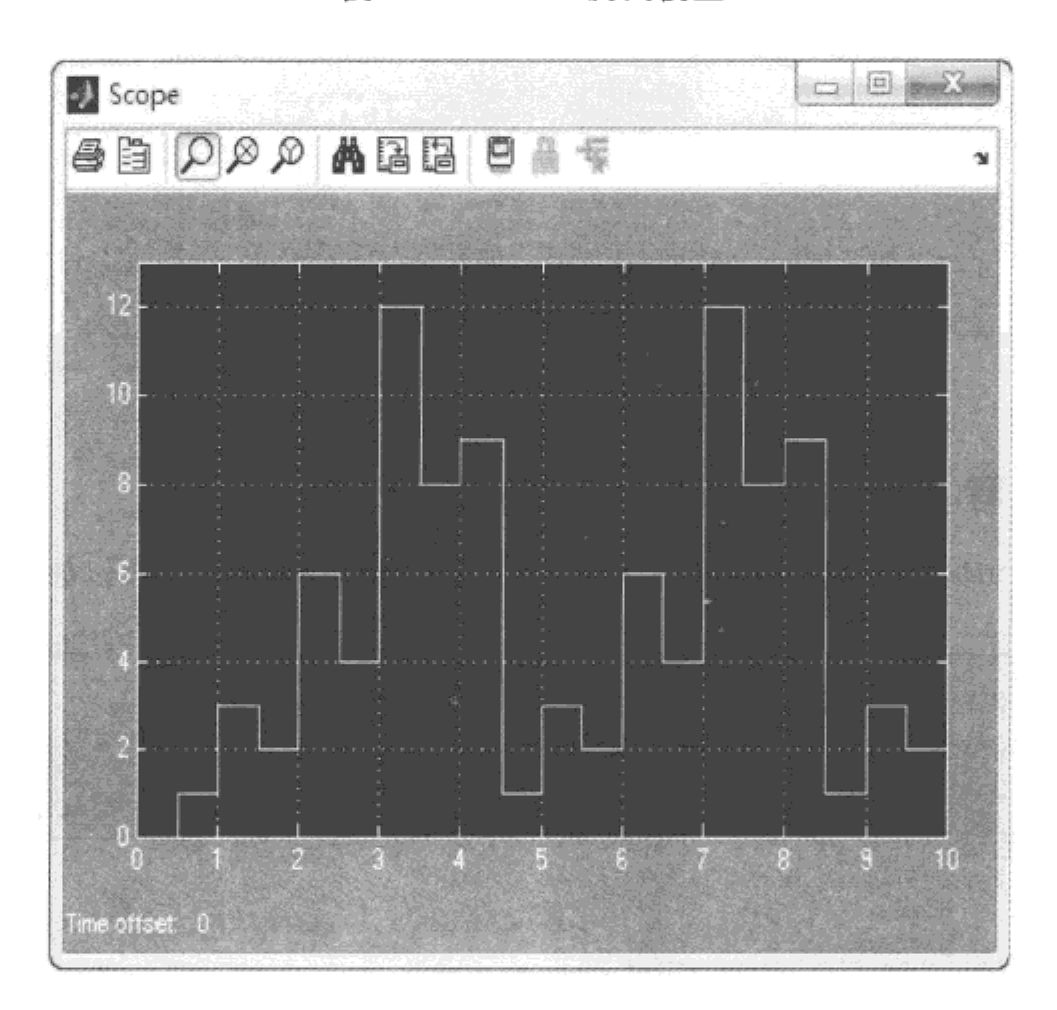


图 8.4.19　仿真结果

8.4.5　自动代码生成

打开模型的参数设置对话框，在 Hardware Implimentation 界面中，设置器件类型为 ARM 7，如图 8.4.20 所示。

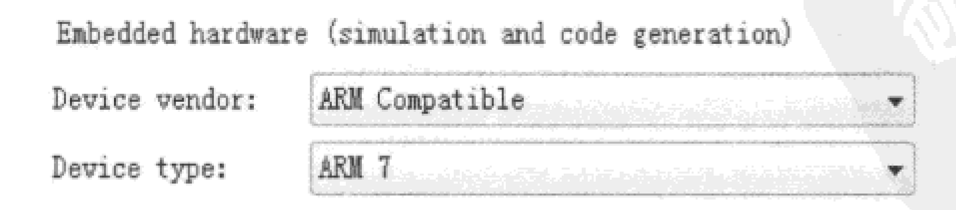


图 8.4.20　选择芯片

单击模型工具栏的按钮，生成代码的报告如图 8.4.21 所示。

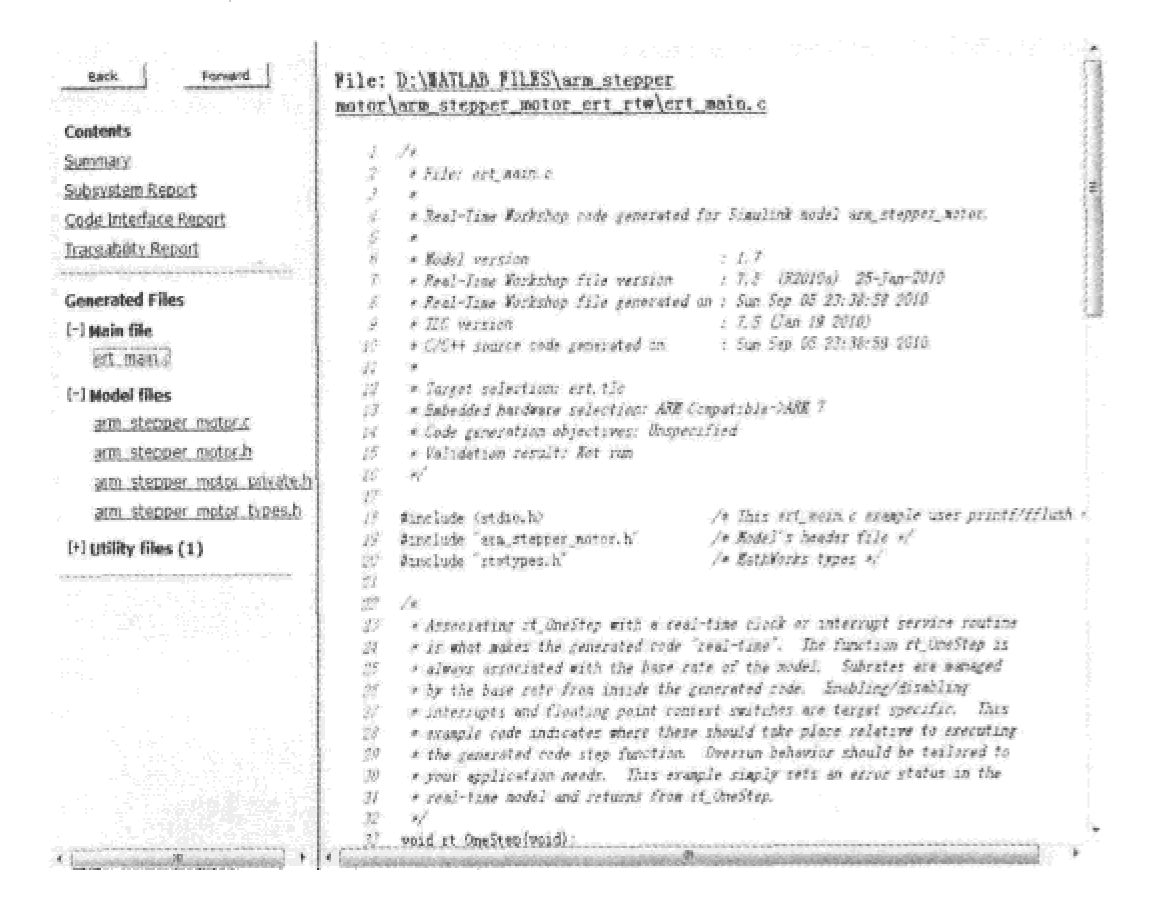


图 8.4.21　代码生成报告

1. 建立 Keil 工程

打开 Keil uVision4，芯片选择 LPC2103，建立工程，并加入生成的代码，如图 8.4.22 所示。

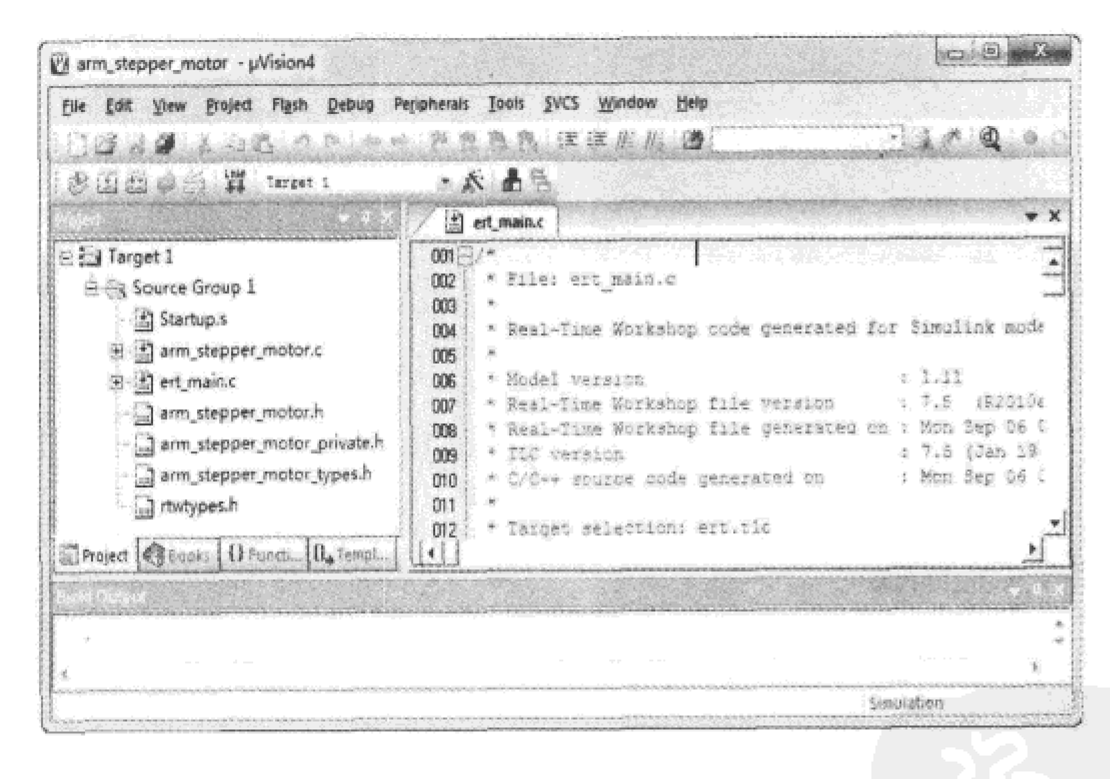
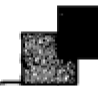


图 8.4.22　Keil 工程

代码并不能直接使用，还需在 ert_main. c 中作如下修改：

```
........
// #include <stdio.h>          /* 在后面的代码中删除了函数 printf/fflush,因此不需要 stdio.h */
#include "arm_stepper_motor.h"            /* Model's header file */
#include "rtwtypes.h"                     /* MathWorks types */
#include <LPC21XX.H>                      /* LPC21XX 头文件 */
// 手动添加 Timer0 的终端服务程序
```



```
void __ irq flag(void)                         //中断服务程序
{
    if(arm_stepper_motor_U.In1 == 0)
    arm_stepper_motor_U.In1 = 1;               //flag 置 1
    else arm_stepper_motor_U.In1 = 0;          //flag 置 0
    T0IR = 0x01;                               //清除中断标志
    VICVectAddr = 0x00;                        //通知 VIC 中断处理结束
    }
// 手动添加 Timer0 的初始化代码
void Timer0Init(void)                          //初始化定时器 0
{
  T0PR = 99;                                   //定时器分频系数设置为 100
  T0MCR = 0x03;                                //匹配通道 0 匹配中断并复位 T0TC
  T0MR0 = 2500;                                //匹配通道 0 的比较值
  T0TCR = 0x03;                                //启动并复位 T0TC
  T0TCR = 0x01;
  //设置定时器 0 中断 IRQ
  VICIntSelect = 0x00;                         //所有中断通道设置为 IRQ 中断
  VICVectCntl0 = 0x24;                         //定时器 0 中断设为最高优先级
  VICVectAddr0 = (uint32_T)flag;               //设置中断服务程序地址向量
  VICIntEnable = 0x00000010;                   //使能定时器 0 中断
}
........
  /* Set model inputs here */
/*将模型输入与硬件相对应*/
  /* Step the model */
  arm_stepper_motor_step();

  /* Get model outputs here */
/*将模型输出与硬件相对应*/
.........
int_T main(int_T argc, const char_T *argv[]);
int_T main(int_T argc, const char_T *argv[])
{
  /* Initialize model */
  arm_stepper_motor_initialize();
  Timer0Init();                                // 调用定时器 0 的初始化函数

 // 删去函数 printf 和 fflush
// printf("Warning: The simulation will run forever. "
//          "Generated ERT main won't simulate model step behavior. "
//          "To change this behavior select the 'MAT-file logging' option.\n");
// fflush((NULL));
  while (rtmGetErrorStatus(arm_stepper_motor_M) == (NULL)) {
    /* Perform other application tasks here */
        rt_OneStep();                          //调用 rt_OneStep()函数
  }
```

```
    /* Disable rt_OneStep() here */

    /* Terminate model */
    arm_stepper_motor_terminate();
    return 0;
}
..........
```

2. 生成 HEX 文件

单击 Keil 工具栏的按钮，或按 Alt+F7 组合键，在 Output 选项卡中，勾选上 Create HEX File 复选框，如图 8.4.23 所示。

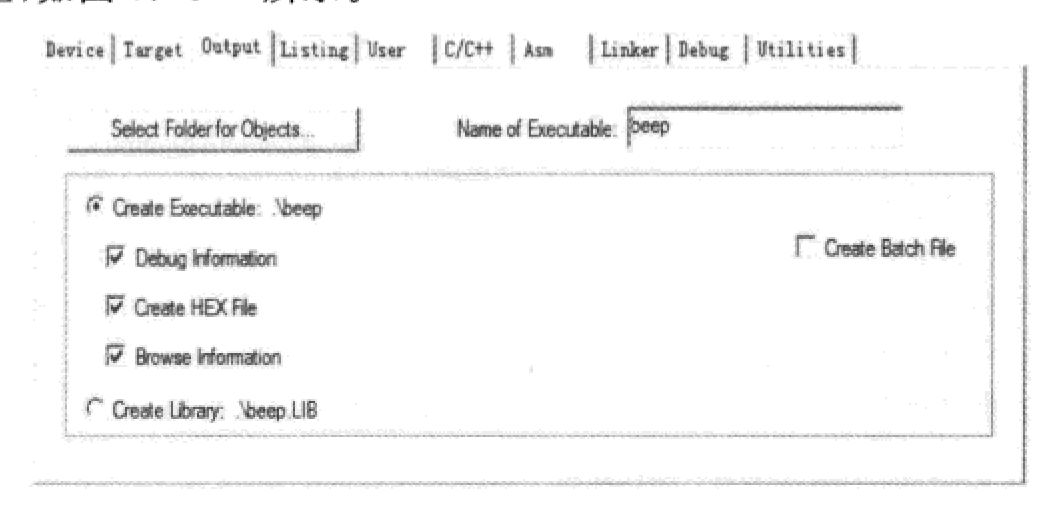


图 8.4.23 设置输出 HEX 文件

再单击工具栏按钮，重编译工程，如图 8.4.24 所示，窗口下部的信息显示已成功生成 HEX 文件。

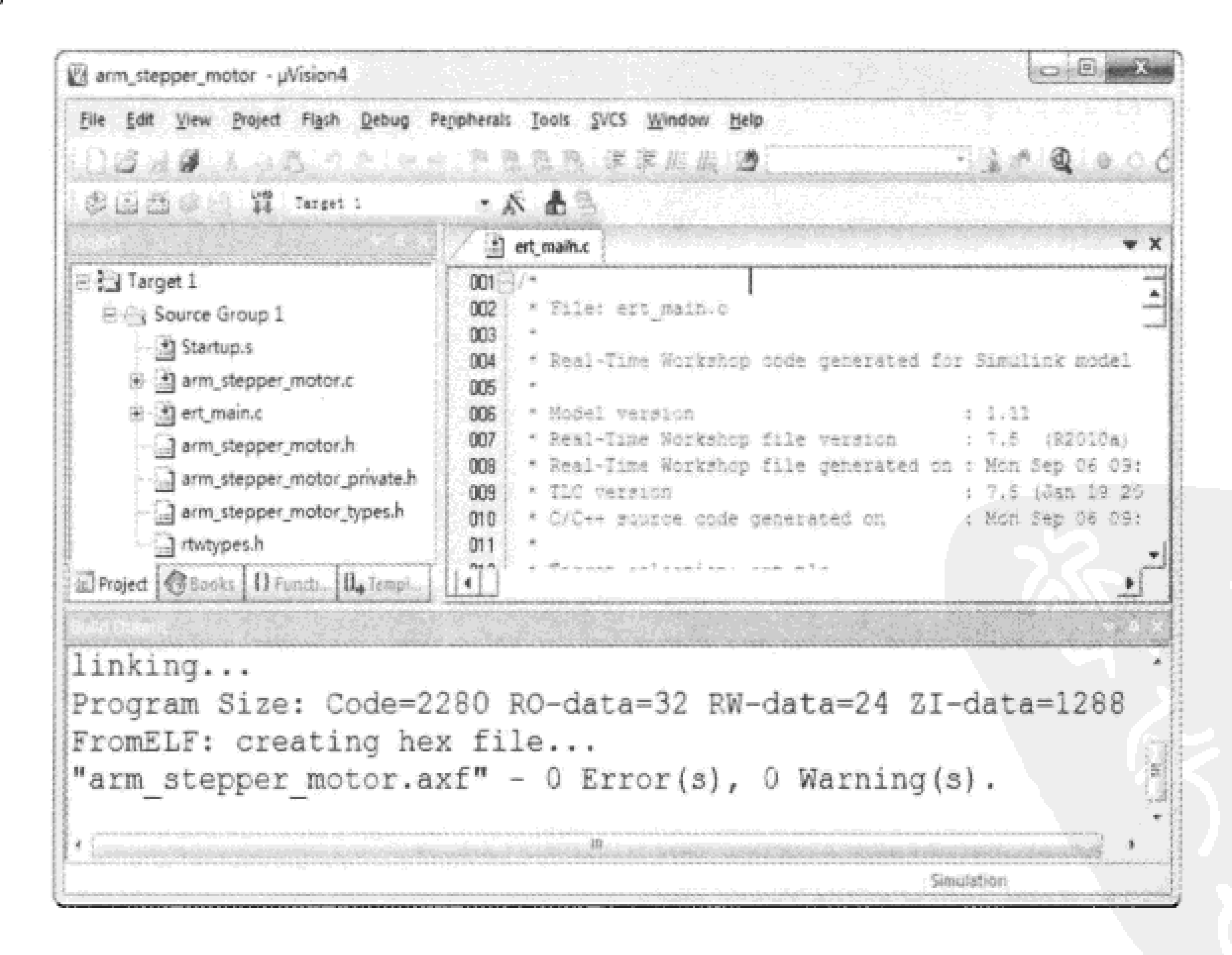
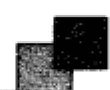


图 8.4.24 编译信息



8.4.6　虚拟硬件测试

建立图 8.4.25 所示的驱动步进电动机模型，ULN2003A 芯片用以驱动步进电动机。添加示波器来监视 ULN2003A 输出的波形是否符合设计逻辑。

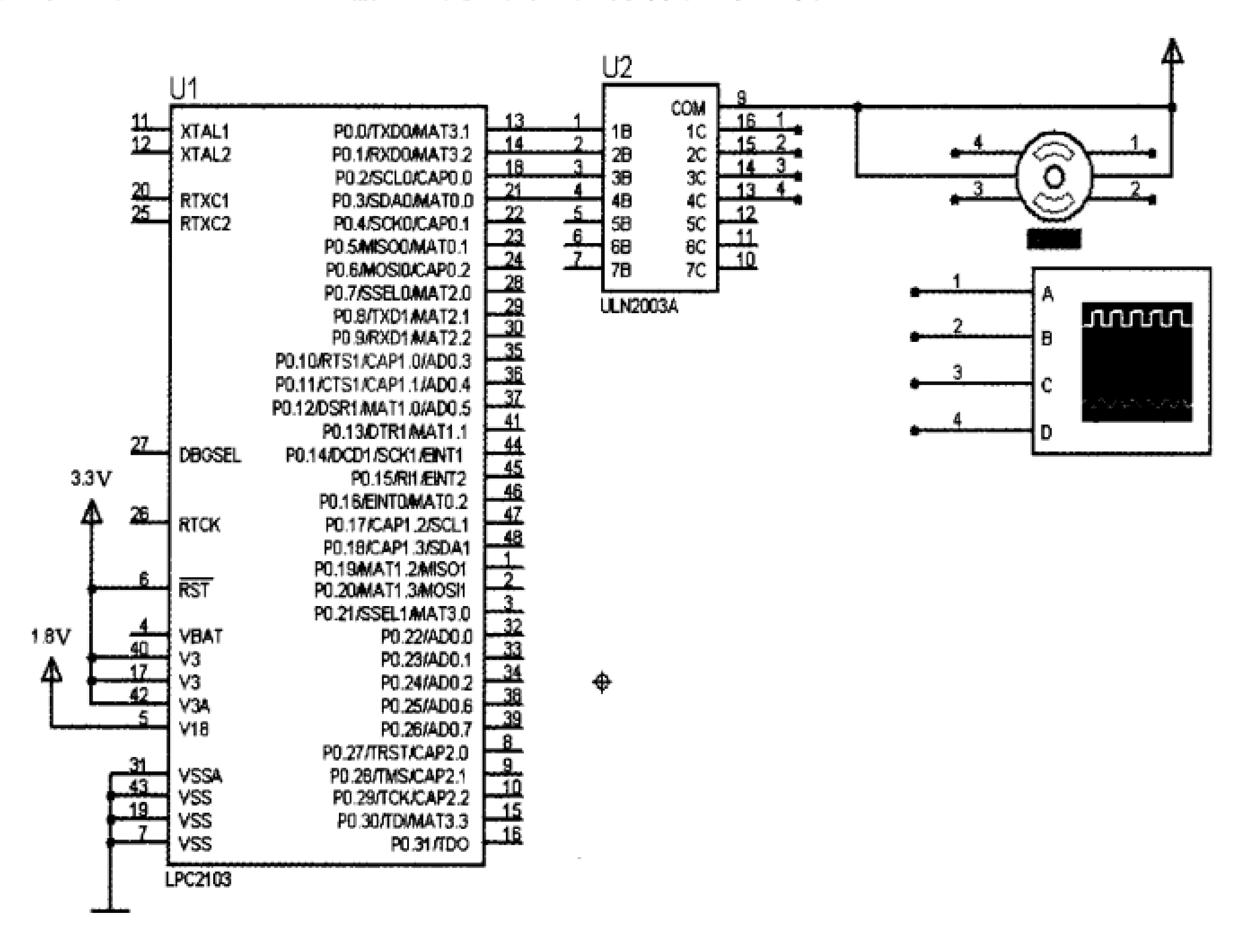


图 8.4.25　Proteus 原理图

打开电动机模块，将其电压设置为 5V，步进角设为 30°，如图 8.4.26 所示。

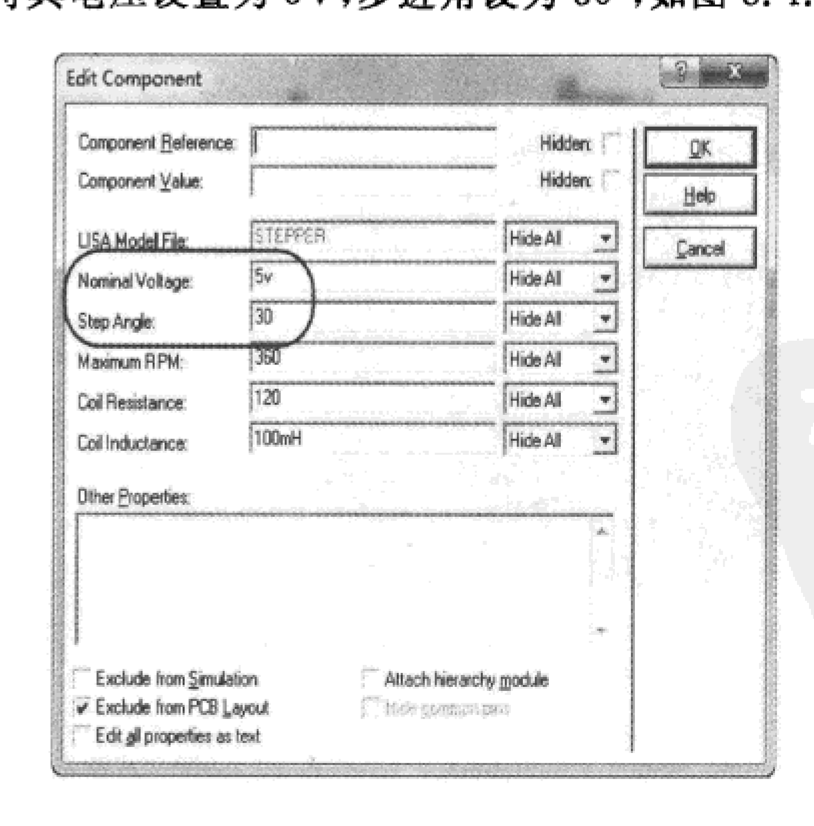


图 8.4.26　设置电动机参数

加载先前生成的 HEX 文件，单击“仿真”按钮。可以看到，电动机正常运行，符合模型预期功能，如图 8.4.27 所示。

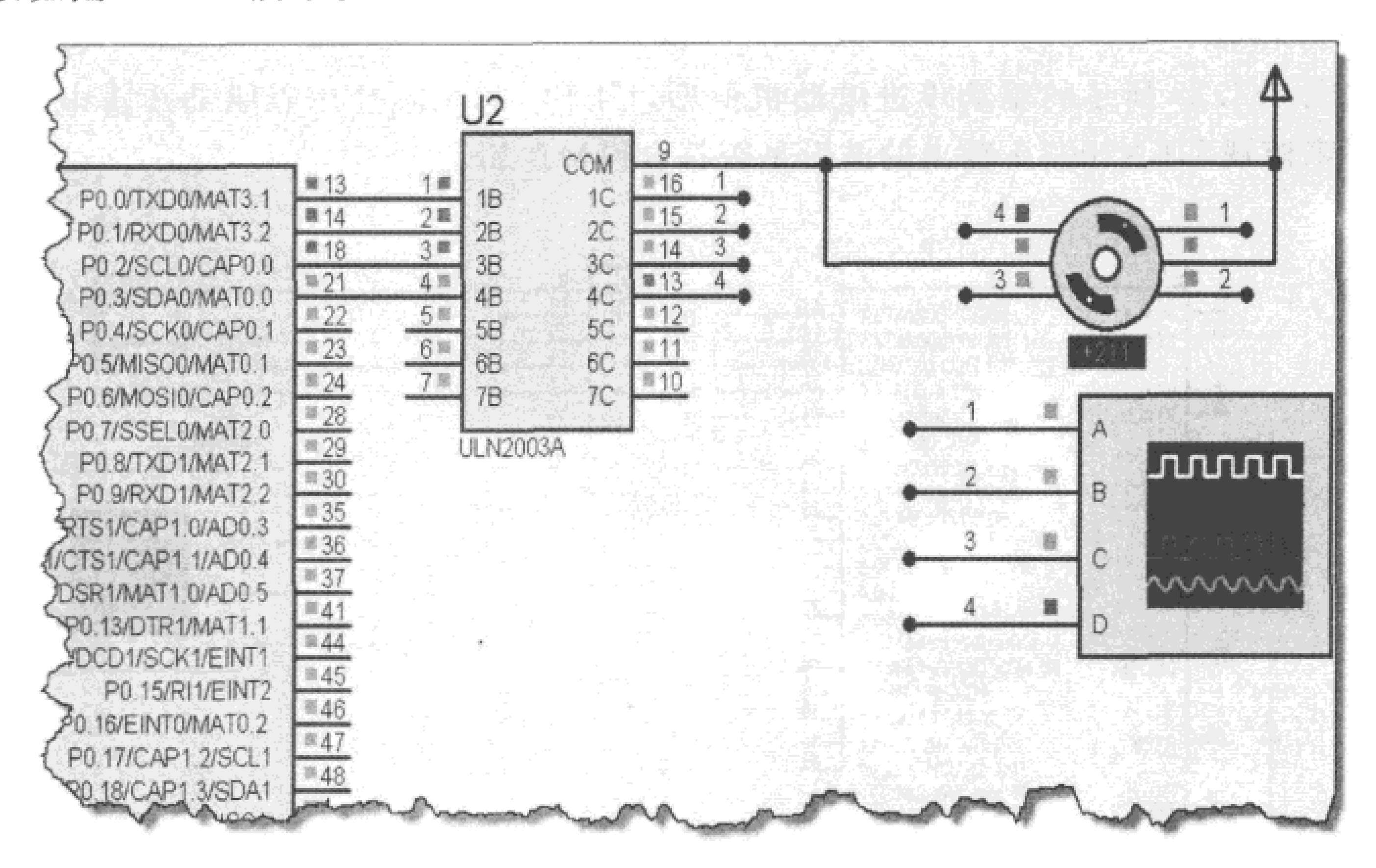


图 8.4.27　仿真结果

由 ULN2003A 模块输出的电平时序如图 8.4.28 所示。

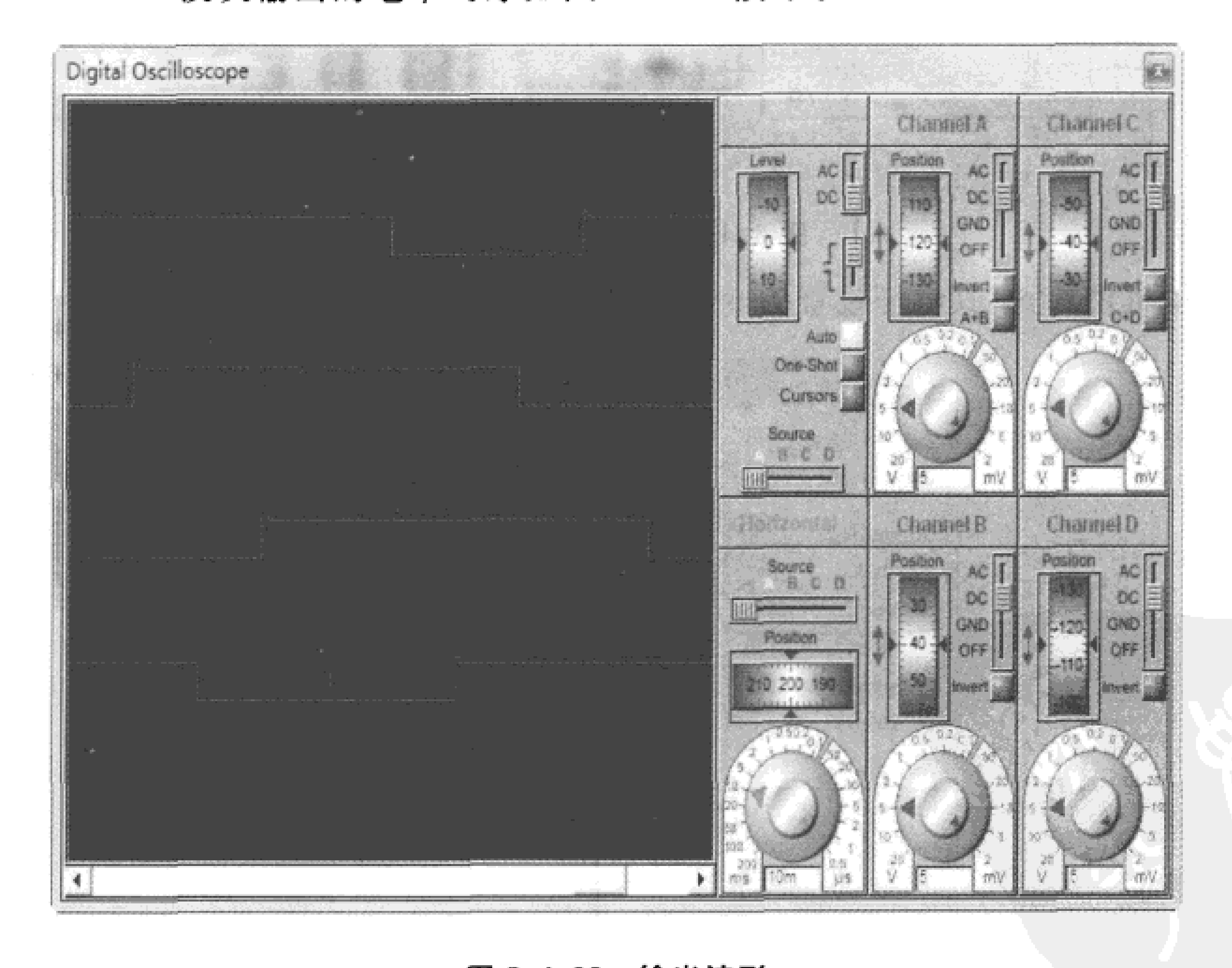
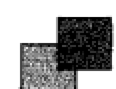


图 8.4.28　输出波形



8.5　无刷电动机的控制

8.5.1　无刷电动机原理简介

目前,电动机可以分为两类:直流电动机和交流电动机。直流电动机是最早出现的电动机,也是最早实现调速的电动机,具有良好的线性调速特性、简单的控制性能、高质高效平滑运转的特性,不过由于电刷和换向器的存在阻碍了它的发展,逐渐被交流电动机所取代。

直流无刷电动机用装有永磁体的转子取代有刷电动机的定子磁极,用逆变器和转子位置传感器组成的电子换向器取代有刷直流电动机的机械换相器和电刷。这样,有效地避免了因电刷引起的火花和噪声等问题,机械特性和调节特性得到了提高,调速范围宽、寿命长、噪声小、不存在换相火花。此外,还具有交流电动机结构简单、运行可靠、维护方便的特点。可用于一般直流电动机无法应用的易燃、易爆环境。

直流无刷电动机的工作离不开电子开关电路,因此由电动机本体、转子位置传感器和电子开关电路三部分组成了直流无刷电动机的控制系统,其原理框图如图 8.5.1 所示,直流电源通过开关电路向电动机定子绕组供电,位置传感器随时检测到转子所处的位置,并根据位置信号来控制开关管的导通和截止,从而自动地控制了哪些绕组通电,哪些绕组断电,实现了电子换相。

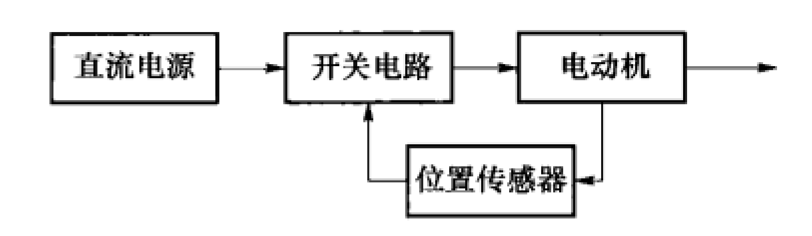


图 8.5.1　控制无刷电动机原理图

下面以一个三相绕组的无刷电动机为例简要介绍其工作原理。图 8.5.2 所示电路为三相全桥式驱动电路,本例对其采用二相通电的方式驱动,即有两个绕阻同时通电。图中包含有 6 个功率管,二极管组成的三相逆变电路,Ha、Hb、Hc 为霍尔元件反馈的转子位置信号。控制电路会根据位置信号决定 6 路 PWM 信号的通断,进而使功率管导通或关断使绕阻按一定顺序导通,驱动电动机连续旋转。

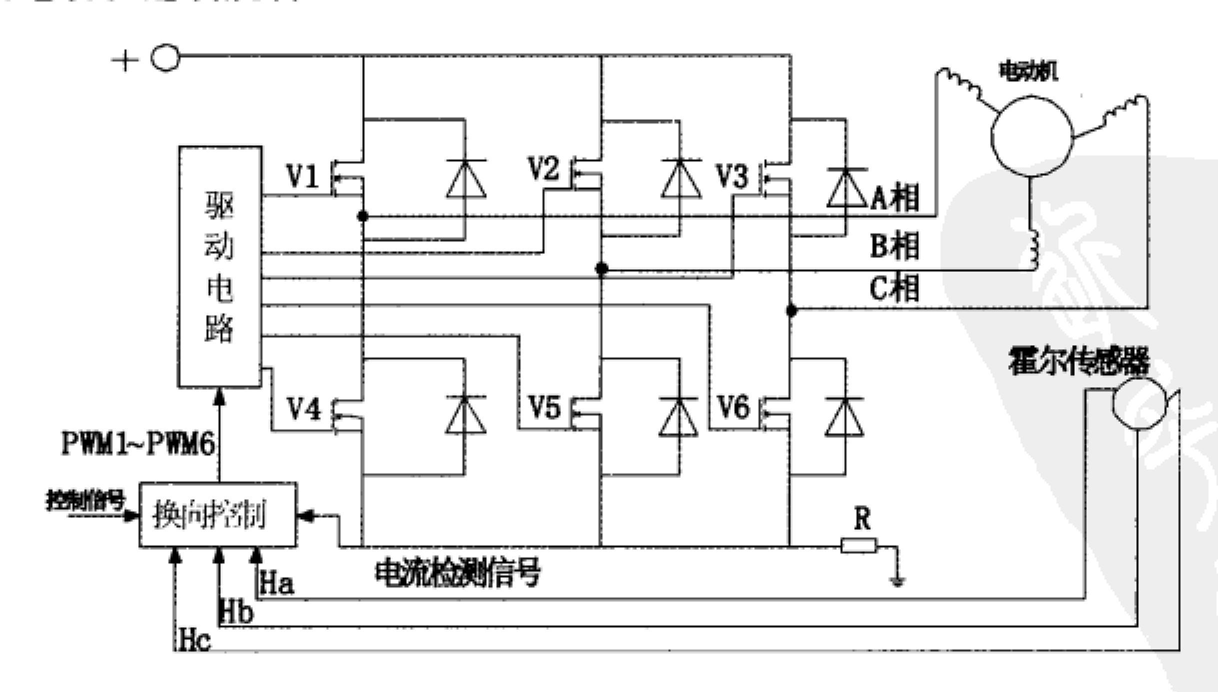


图 8.5.2　驱动电路原理图

当采用二相导通方式驱动电动机时，功率管的导通或关断情况每经过 1/6 周期即 60°。在直流无刷电动机的内部嵌有 3 个霍尔位置传感器，它们在空间上相差 120 °。由于电动机的转子是永磁体，当它在转动的时候，其磁场将发生变化形成旋转磁场，每个霍尔传感器都会产生 180°脉宽的输出信号。

假设当前功率管 V1，V6 导通，则电流从 A 相流入电动机，从 C 相流出电动机，由电流经绕阻产生的磁场方向为 A，-C，如图 8.5.3 所示。由 A 和-C 的合磁场长生的转矩使转子转动到 AC 位置。转子的转动使霍尔传感器的输出发生变化，控制电路会据此调整功率管的导通情况，将 V6 关断，V5 导通。这时，电流从 A 相流入电动机，从 B 相流出电动机，有电流经绕阻产生的磁场方向为 A，-B，如图 8.5.4 所示。由 A 和-B 的合磁场长生的转矩使转子转动到 AB 位置。同样地，霍尔器件又会输出一个不同的值，控制电路做出相应的处理，完成一个完整的换相周期。

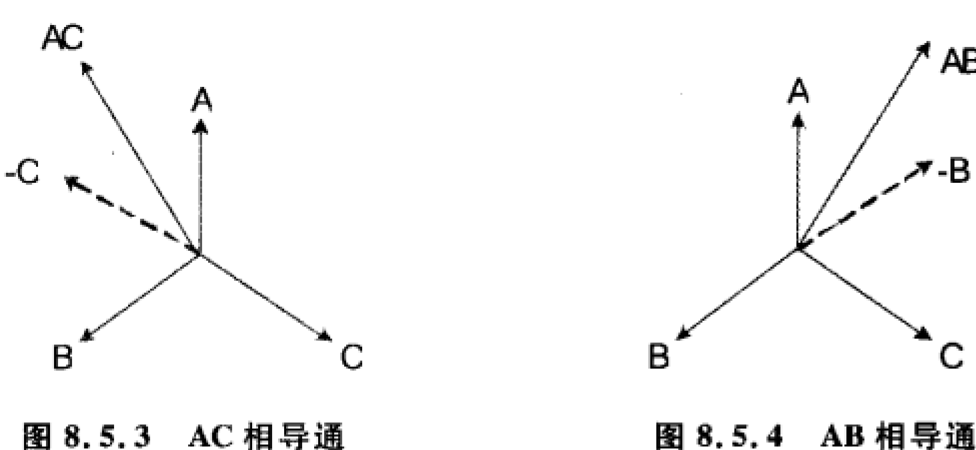


图 8.5.3　AC 相导通　　　　图 8.5.4　AB 相导通

8.5.2　TASKING IDE FOR ARM

TASKING VX-toolset for ARM 是 Altum 公司推出的一款 ARM 编译器，能够完成从 Edit、Make 到 Debug 等一整套开发工序，并针对 ARM 芯片做过优化设计，可有效提高其效率。

如果用户没有这款软件，则可在 http://www.tasking.com/products/trials-and-demos.shtml，页面中下载其试用版本，单击 ARM VX-Toolset Trial Version 填入必要注册信息后即可下载，如图 8.5.5 所示。

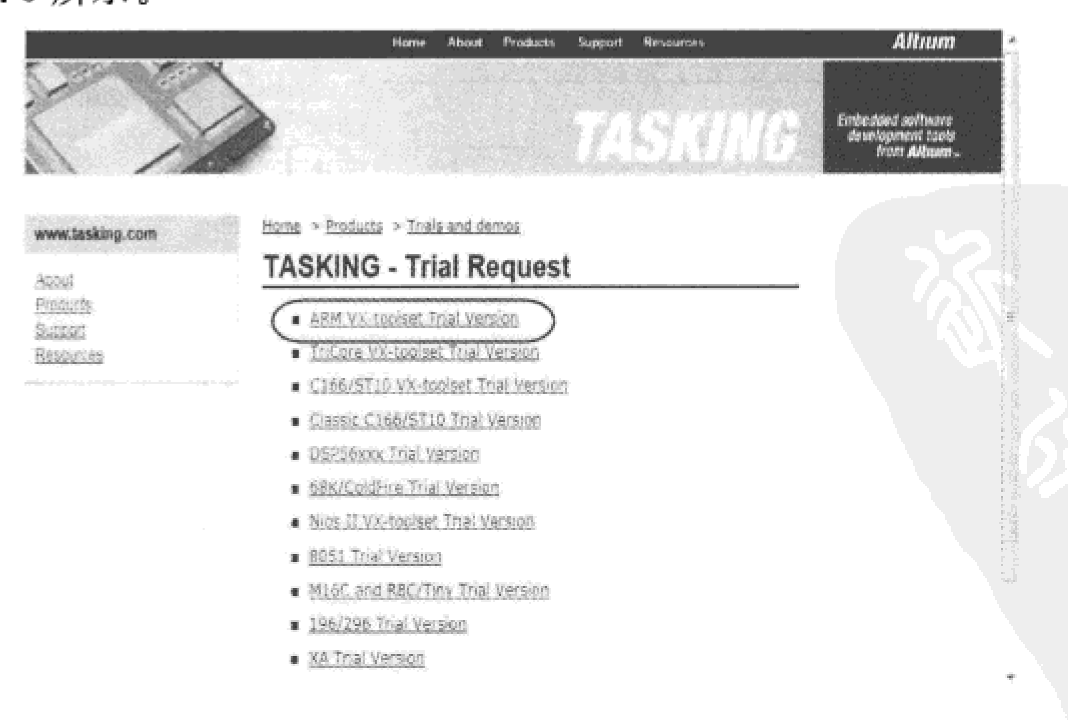
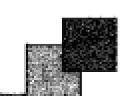


图 8.5.5　TASKING 网页



TASKING VX-toolset for ARM 编译环境针对 ARM 芯片进行了优化，效率较高，可以独立使用，但由于此试用版本与 MATLAB 所支持的 TASKING 版本号并不相同，因此若要和 MATLAB 联合工作，用于自动代码生成，还需要更换部分文件，用户可登录北京航空航天大学出版社“下载中心”下载并替换原文件。

注意：此方法仅用于学习本书之用，如果读者想用于项目开发，还请安装 MathWorks 公司指定的 TASKING EDE 正版软件。

8.5.3　无刷电动机控制模型

根据上述原理简介可知，无刷电动机由一组 PWM 信号驱动。PWM 信号按霍尔元件传送的位置信号决定其通断状态，以驱动电动机连续旋转；而 PWM 信号占空比可用于调节电动机转速。在 Init 状态中，为模型设置 PWM1～PWM6 六路 PWM 信号，并以按 key 键的值控制电动机的开关，由此可作无刷电动机的状态图，如图 8.5.6～图 8.5.8 所示。

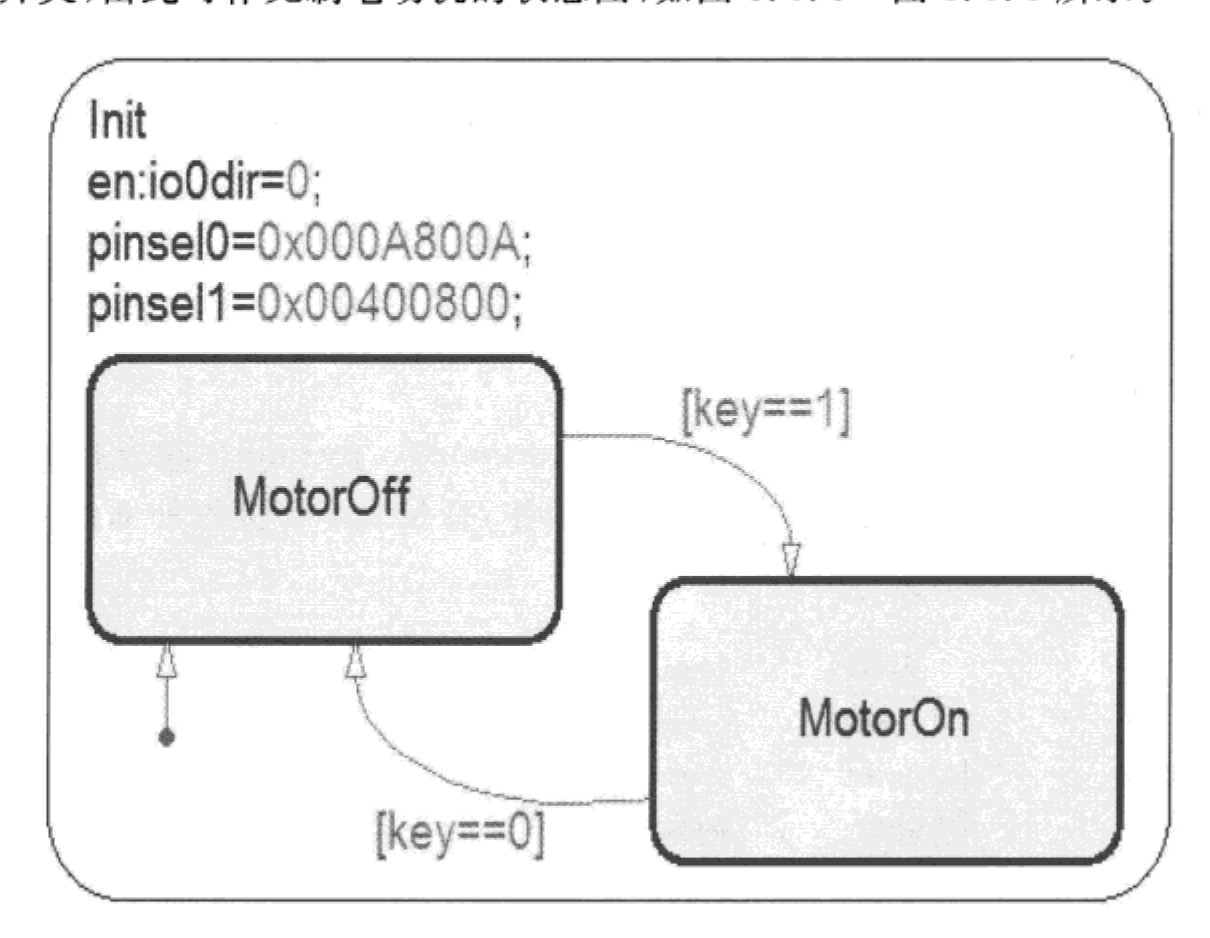


图 8.5.6　无刷电动机控制模型

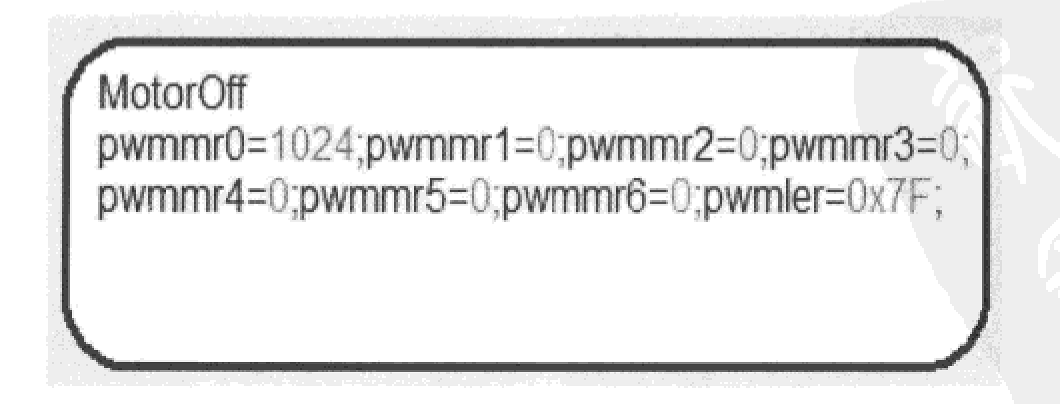


图 8.5.7　MotorOff 子状态

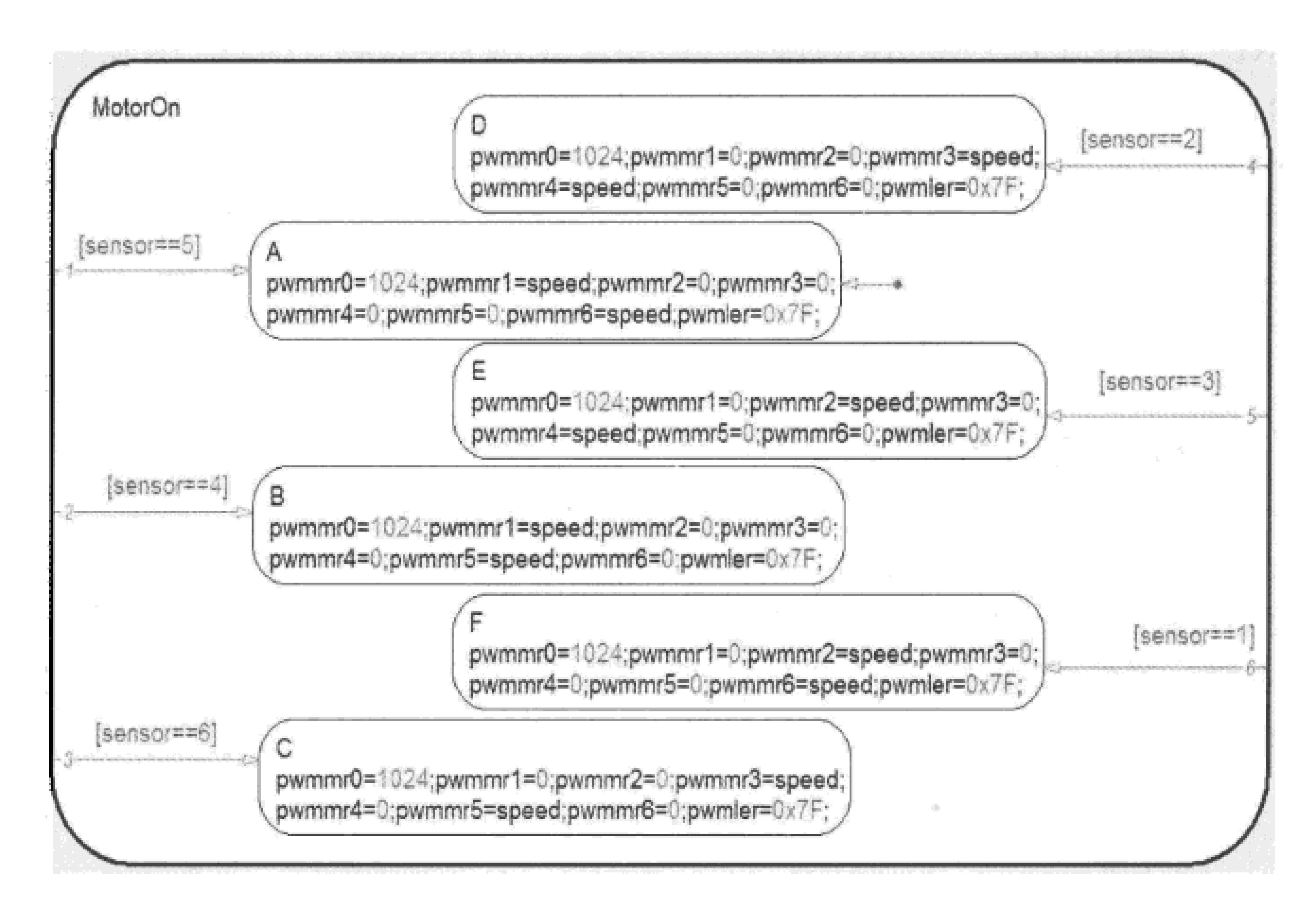


图 8.5.8　MotorOn 子状态

在 MotorOff 子状态中，将六路 PWM 信号的占空比调至 0，以达到关闭电动机的作用。

在 MotorOn 子状态中，模型接收霍尔元件传送回的电动机转子位置信号，并以此判断 PWM 信号的通断。

其中 pinsel0，pinsel1，io0dir 为芯片设置位，pwmmr0～pwmmr6 联合控制 PWM 输出，sensor 表示霍尔器件的值，key 控制电动机是否工作，变量 speed 接收外部的控制信号，调节 PWM 占空比，实现电动机调速，如图 8.5.9 所示。

Name	Scope	Port	Resolve Signal	DataType	Size
pinsel0	Output	1		double	
pinsel1	Output	2		double	
io0dir	Output	3		double	
pwmmr0	Output	4		double	
sensor	Input	1		double	
key	Input	2		double	
speed	Input	3		double	
pwmmr1	Output	5		double	
pwmmr2	Output	6		double	
pwmmr3	Output	7		double	
pwmmr4	Output	8		double	
pwmmr5	Output	9		double	
pwmmr6	Output	10		double	
pwmler	Output	11		double	

图 8.5.9　无刷电动机控制模型数据列表



8.5.4　无刷电动机功能验证模型

完成 Stateflow 状态图之后，在 Simulink 模块库中找到图 8.5.10～图 8.5.12 所示模块，并按图 8.5.13 所示连接。

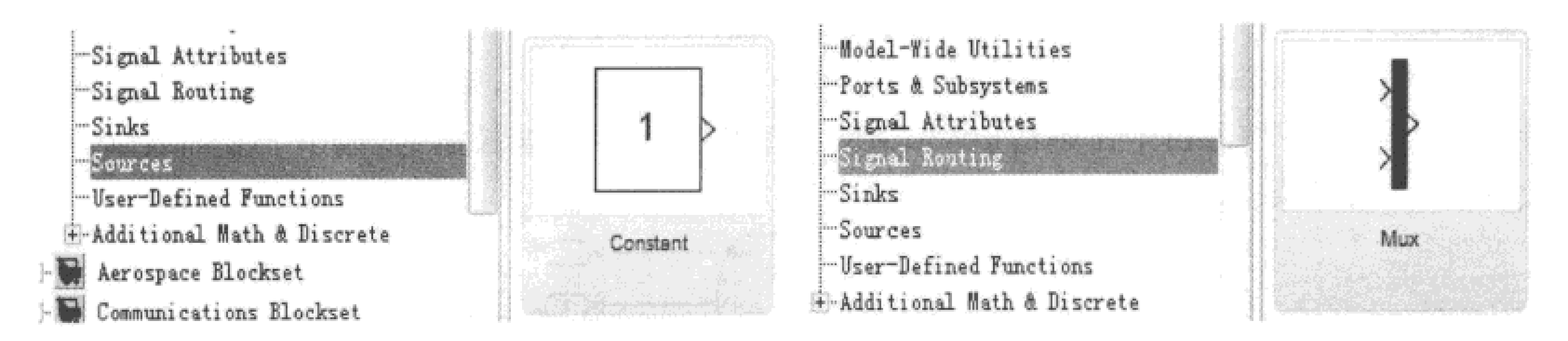


图 8.5.10　常数模块　　　　图 8.5.11　集线器模块

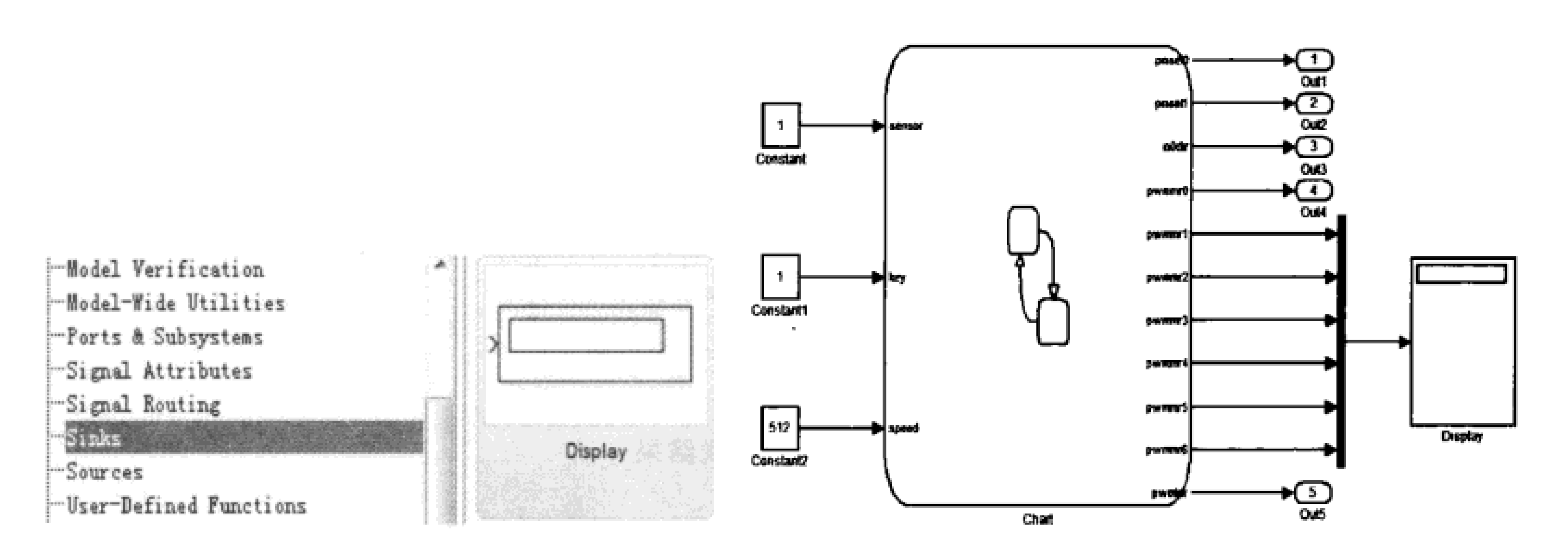


图 8.5.12　显示模块　　　　图 8.5.13　功能验证模型

将 Display 模块的维度设置为 6，如图 8.5.14 所示。

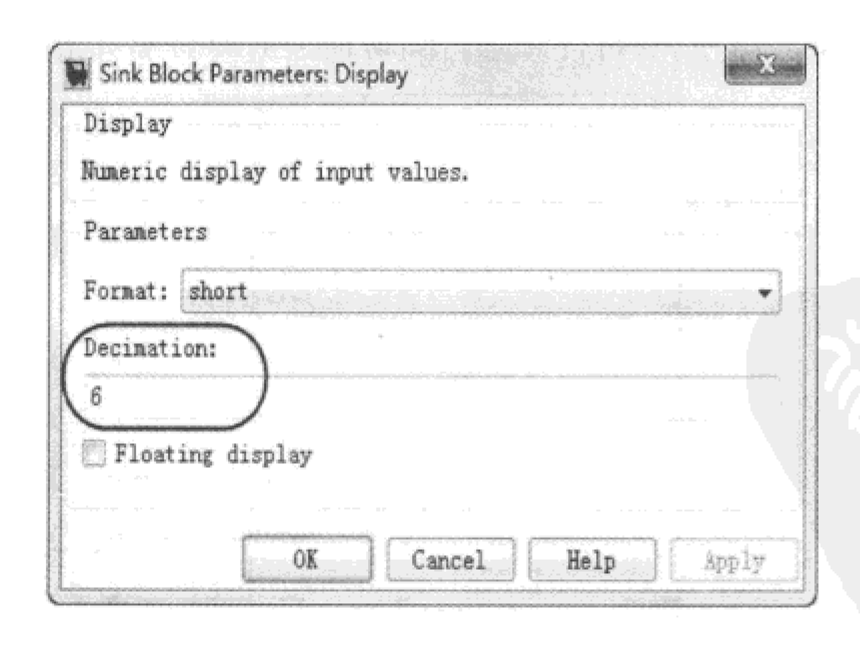


图 8.5.14　设置 Display 模块参数

选择模型主窗口的菜单项 Simulation→Configuration Parameters...，打开模型参数对话框，在 Solver 面板中，设置求解器为定步长离散求解器，步长为 0.01，如图 8.5.15 所示。

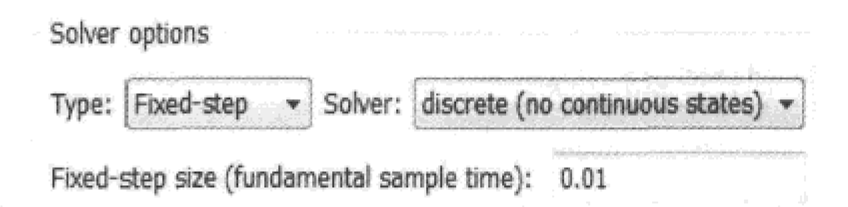


图 8.5.15 求解器设置

当 Key＝1,电动机处于打开状态时,若霍尔传感器状态为 1,则第 2 和第 6 路 PWM 信号导通,输出 512。信号占空比是由 PWMMR0～ PWMMR6 联合控制的,输出 512 即表示占空比为 1∶1,如图 8.5.16 所示。

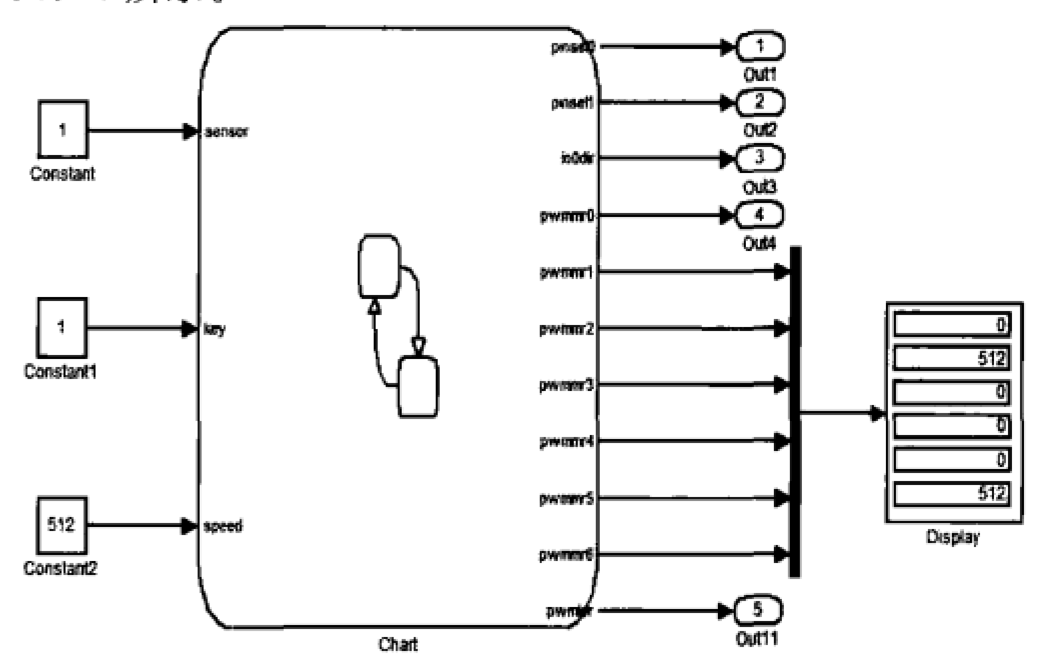


图 8.5.16 仿真结果

8.5.5 软件在环测试

在模块库 Simulink / Ports &Subsystems 中找到输入模块,图 8.5.13 的 Constant 模块,用输出模块替换 mux 模块和 display 模块,并将这些端口的数据类型修改为 uint32,如图 8.5.17 所示。

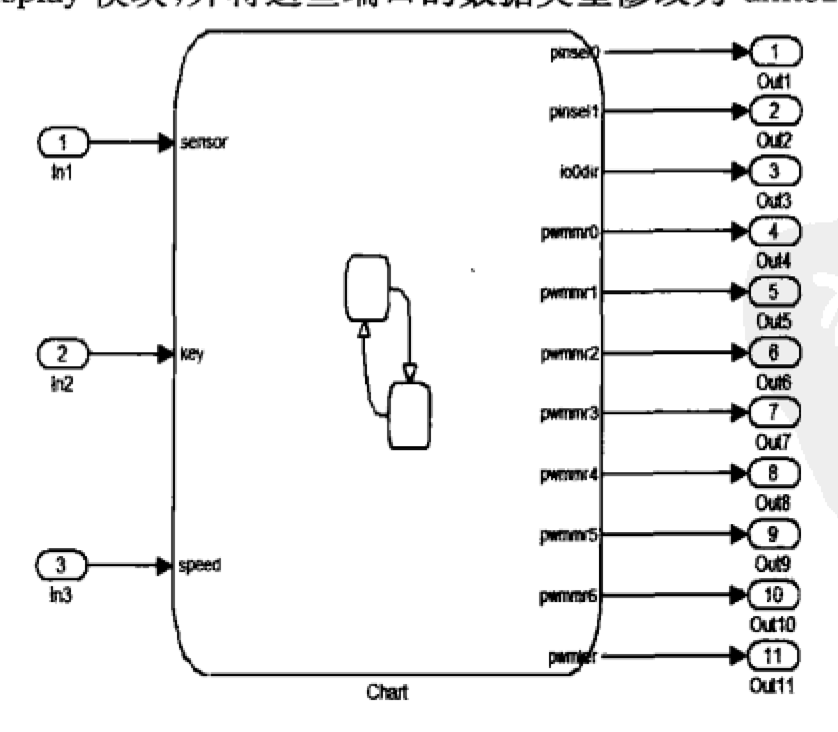
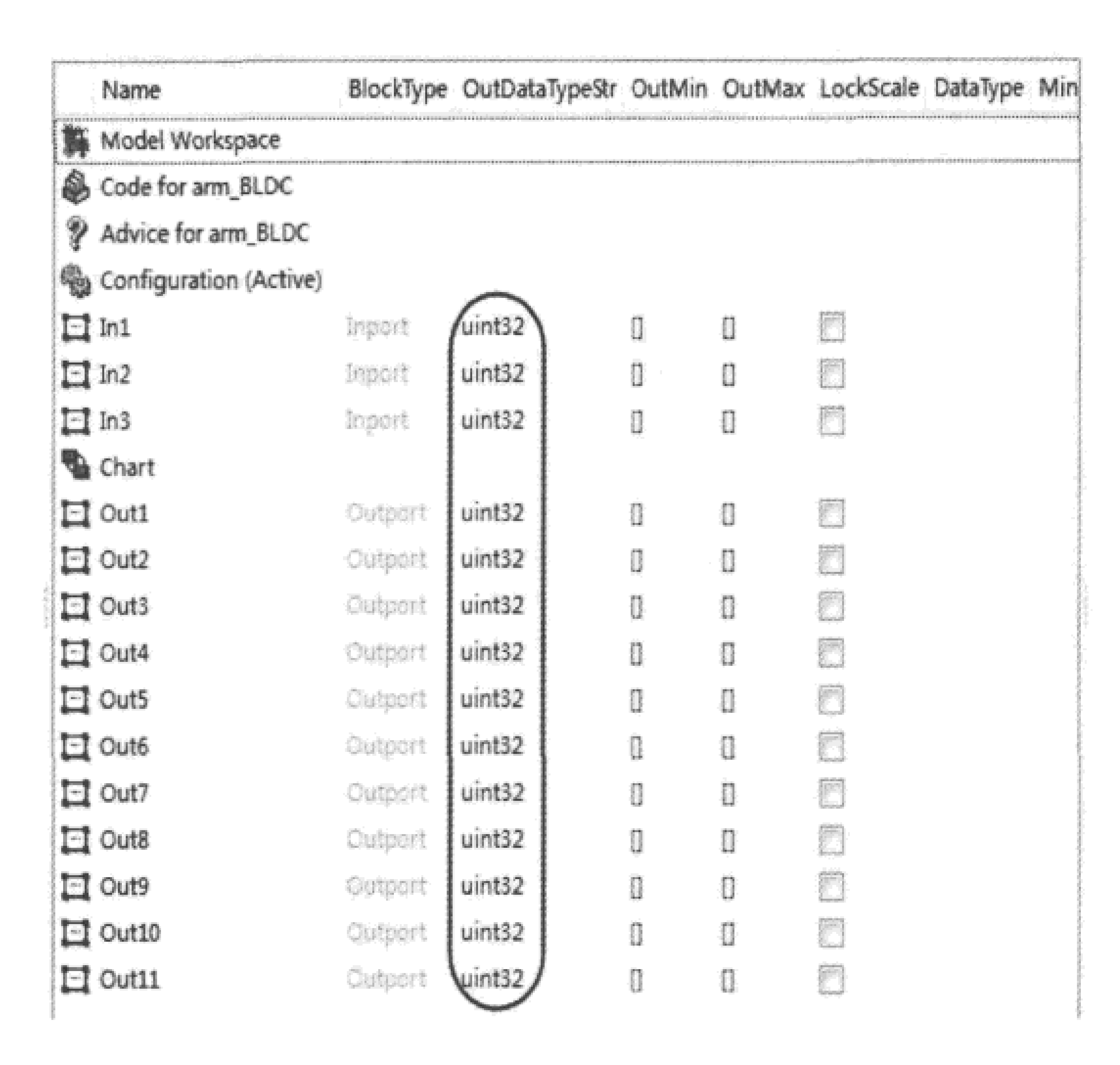


图 8.5.17 代码模型

另外还需要确保 Stateflow 模型中数据的类型也应为 uint32，如图 8.5.18、图 8.5.19 所示。



Name	BlockType	OutDataTypeStr	OutMin	OutMax	LockScale	DataType	Min
Model Workspace							
Code for arm_BLDC							
Advice for arm_BLDC							
Configuration (Active)							
In1	Inport	uint32	[]	[]			
In2	Inport	uint32	[]	[]			
In3	Inport	uint32	[]	[]			
Chart							
Out1	Outport	uint32	[]	[]			
Out2	Outport	uint32	[]	[]			
Out3	Outport	uint32	[]	[]			
Out4	Outport	uint32	[]	[]			
Out5	Outport	uint32	[]	[]			
Out6	Outport	uint32	[]	[]			
Out7	Outport	uint32	[]	[]			
Out8	Outport	uint32	[]	[]			
Out9	Outport	uint32	[]	[]			
Out10	Outport	uint32	[]	[]			
Out11	Outport	uint32	[]	[]			

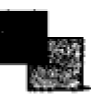
图 8.5.18　修改模型端口数据类型

Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType
pinsel0	Out...	1		uint32			unknown
pinsel1	Out...	2		uint32			unknown
io0dir	Out...	3		uint32			unknown
pwmmr0	Out...	4		uint32			unknown
sensor	Input	1		uint32			unknown
key	Input	2		uint32			unknown
speed	Input	3		uint32			unknown
pwmmr1	Out...	5		uint32			unknown
pwmmr2	Out...	6		uint32			unknown
pwmmr3	Out...	7		uint32			unknown
pwmmr4	Out...	8		uint32			unknown
pwmmr5	Out...	9		uint32			unknown
pwmmr6	Out...	10		uint32			unknown
pwmler	Out...	11		uint32			unknown

图 8.5.19　修改模型内部数据类型

在 Report 界面中，勾选所有复选框，便于后期检查及跟踪，如图 8.5.20 所示。

打开模型的参数设置对话框，在 Real-Time Workshop→SIL and PIL Verification 界面中，勾选 Create SIL block 复选框，如图 8.5.21 所示。



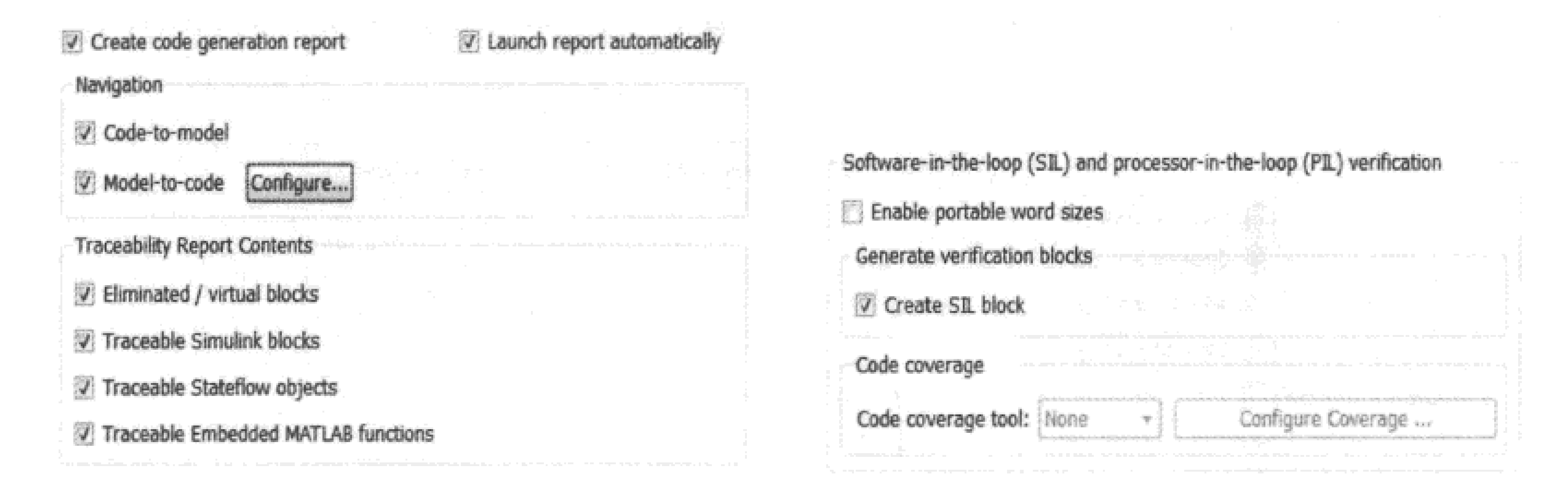


图 8.5.20 报告设置界面　　　图 8.5.21 SIL 设置

在 Real-Time Workshop 界面中，设置 TLC 文件为 ert.tlc，如图 8.5.22 所示。

单击模型工具栏的按钮，生成 SIL 模块，如图 8.5.23 所示。

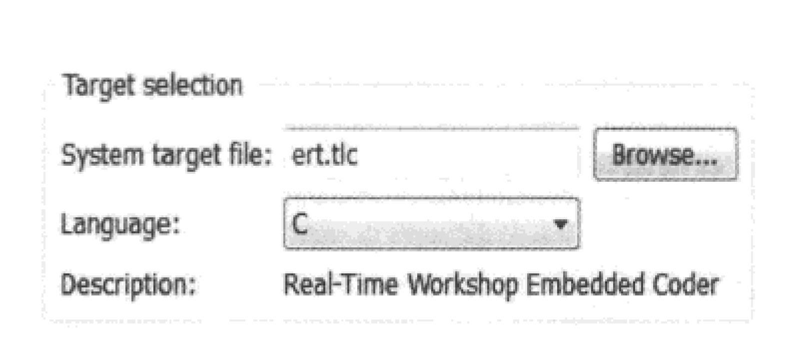


图 8.5.22 设置 TLC

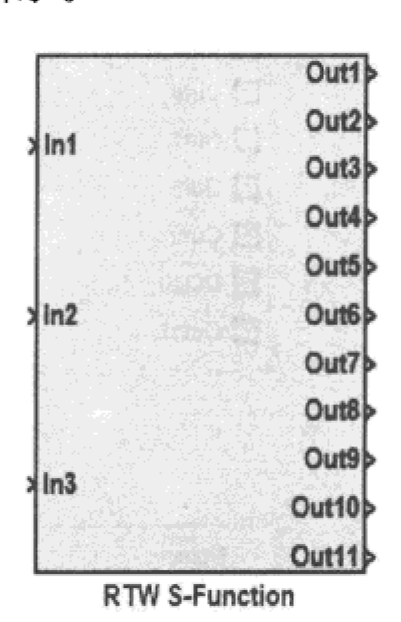


图 8.5.23 SIL 模块

如图 8.5.16 所示，以 SIL 模块替换 Stateflow 模块，重建图 8.5.24 所示的验证模型。由于 SIL 模块是根据定点模型建立的，因此各端口间加入了数据类型转换模块。

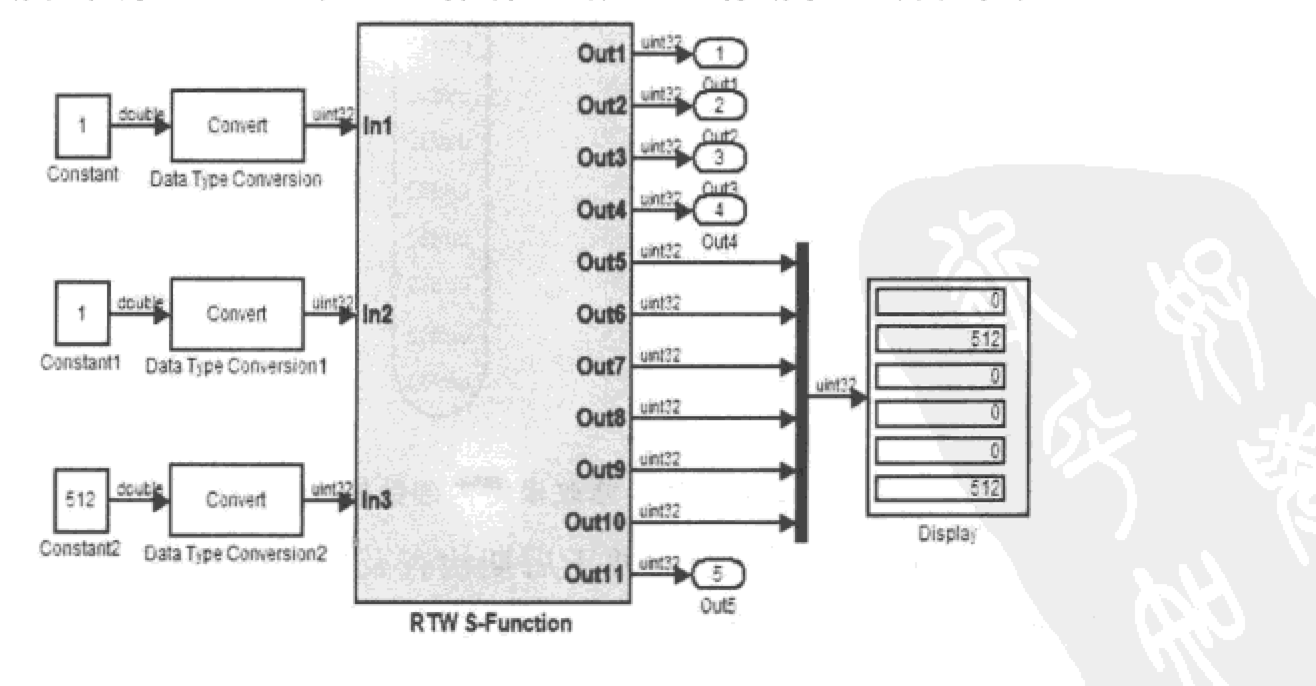
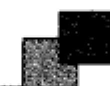


图 8.5.24 SIL 测试模型



可见，该模型的运行结果与图 8.5.16 的结果是一致的。在 Display 模块的第 2、6 行输出了占空比为 1∶1 的 PWM 信号。

8.5.6 编写驱动代码

无刷直流电动机需要一组 PWM 信号驱动开关电路，进而根据霍尔传感器的位置信号有序换相，使电动机能够持续旋转。这里采用 ARM 芯片上集成的 PWM 模块来产生 PWM 信号，因此需要为其添加以下初始化代码：

```
#include"LPC2124.h"
void PWM_Init(void) //初始化 LPC2124 芯片的 PWM 功能
 {
 PWMPR  =  1200; // 设置分频系数
 PWMMR0  =  1024; // PWMMR0～PWMMR6 控制初始占空比
 PWMMR1 = PWMMR2 = PWMMR3 = PWMMR4 = PWMMR5 = PWMMR6 = 0;
 PWMMCR  =  0x00000002; // 重置 MR0 时钟
 PWMPCR  =  0x7E00; // 使能 PWM1～PWM6 输出
 PWMLER  =  0x7F; // 使能 PWM0～ PWM6 锁存
 PWMTCR  =  0x09; // 使能 PWM 模式并启动定时器
 }
```

为了使电动机具备调速功能，本例采用电位器改变输入电压值，进而控制 PWM 信号的占空比的方式实现。因此需要添加 A/D 转换相关代码：

```
#include"LPC2124.h"
void AD_Read(void)
{
     ADCR  =  (ADCR&0x00FFFF00)|0x01|(1 << 24);     // 设置通道 1,并进行第一次转换
     while( (ADDR&0x80000000)== 0 );                // 等待转换结束
     ADCR &=  ～0x01000000;                         // 停止转换
}
```

8.5.7 自动代码生成

添加 TASKING IDE FOR ARM

选择模型菜单项 Tools→Utilities for Use with TASKING(R) IDE→Add Embedded IDE Link Configuartion to Model...，为模型添加 IDE Link 选项。

选择模型菜单项 Tools→Utilities for Use with TASKING(R) IDE→Target Preferences，选择 ARM 选项，设置其目标选项，如图 8.5.25 所示。

指定 xfwarm.exe、carm.dol、ede.exe 这三个文件的位置，与安装路径有关。例如 D:\Program Files\TASKING\carm v3.0r3\...，指定完毕后单击 OK 按钮，如图 8.5.26 所示。

打开模型的参数设置对话框，在 Hardware Implimentation 界面中，设置器件类型为 ARM 7，如图 8.5.27 所示。

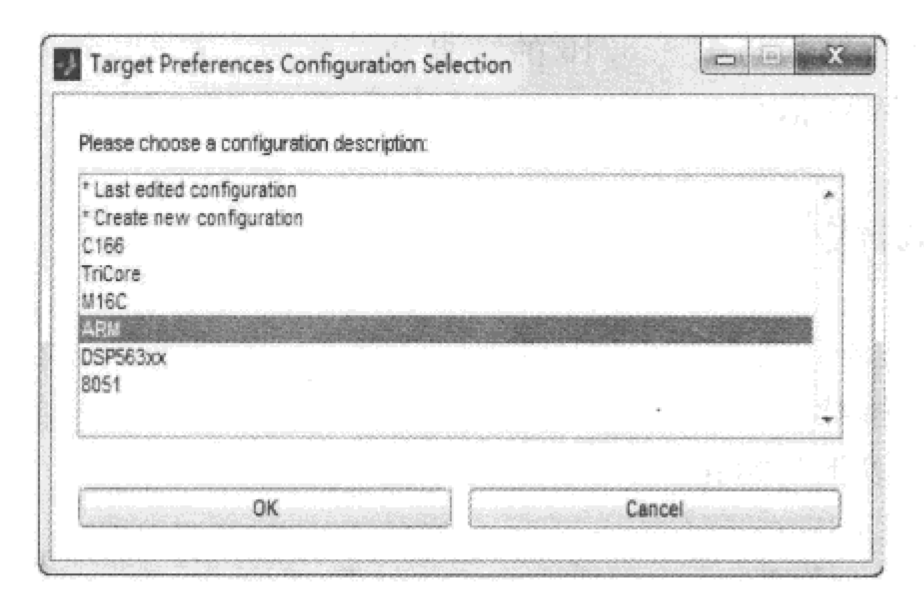


图 8.5.25　目标选择

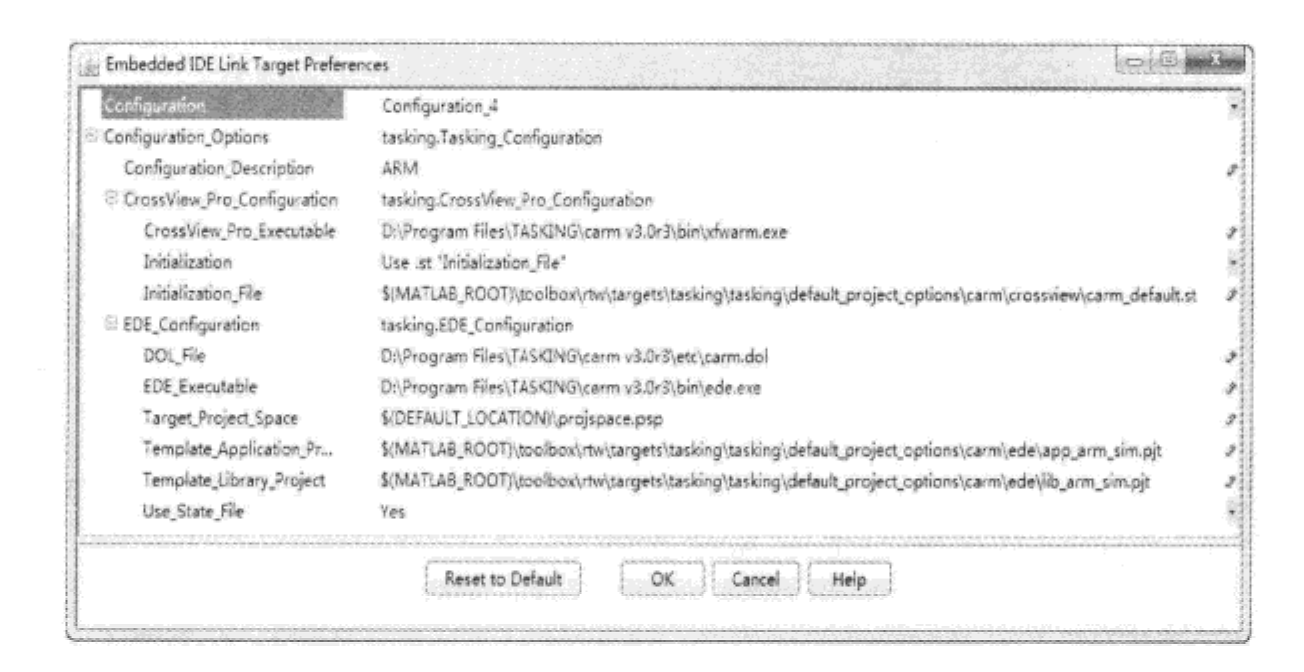


图 8.5.26　IDE 配置

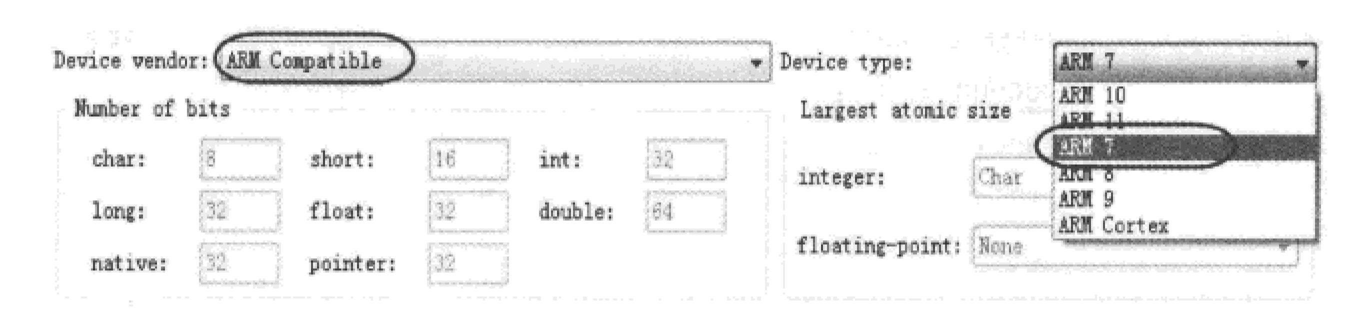


图 8.5.27　选择芯片

在 Embedded IDE Link 界面中，设置器件为 ARM，如图 8.5.28 所示。

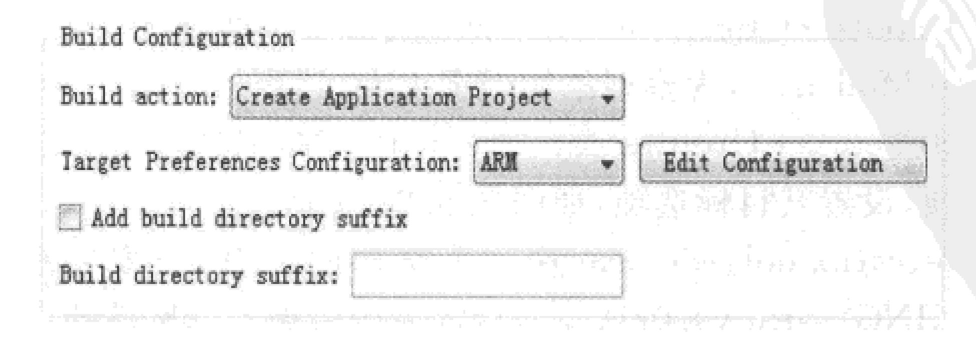
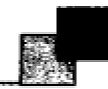


图 8.5.28　Embedded IDE Link 界面设置

在 Real-Time Workshop→Custom Code 面板中的 Include Directories 文本框，输入驱动



代码所在的目录，如 D:\MATLAB FILES\ARM_BLDC_test(图 8.5.29)；Source Files 文本框，输入驱动代码文件名，如 D:\MATLAB FILES\ARM_BLDC_test\PWM_Init.c (图 8.5.30)；在 Library Files 选项，填入库文件名，如 D:\MATLAB FILES\ARM_BLDC_test\LPC2124.h(图 8.5.31)。目录与文件路径，必须用半角双引号包围。

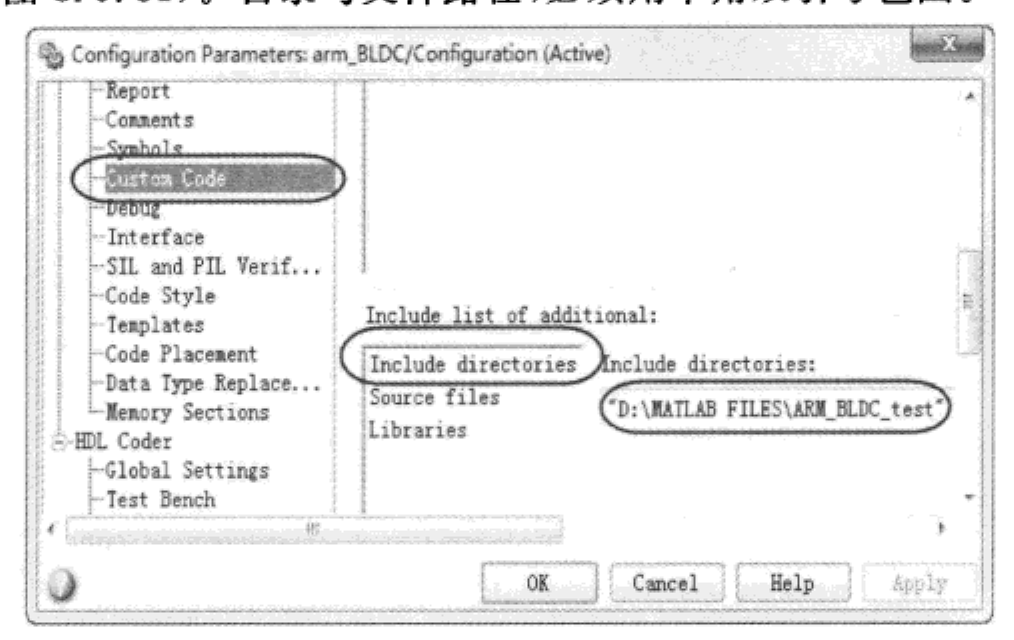


图 8.5.29　设置代码目录

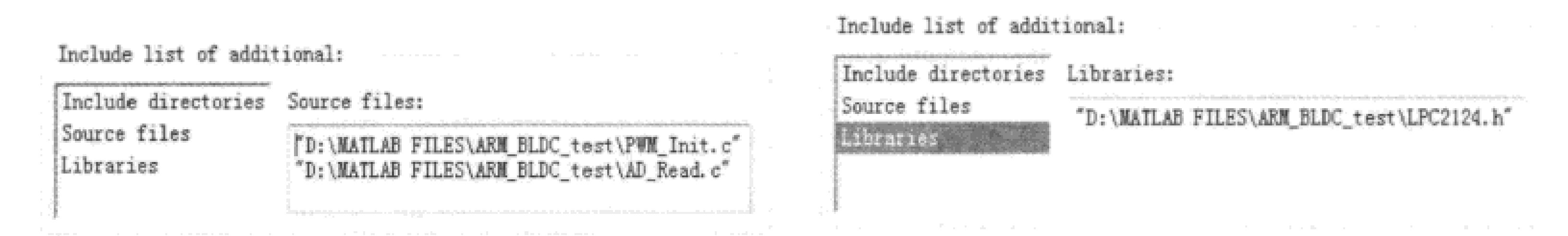


图 8.5.30　指定原代码　　　　图 8.5.31　指定库文件

单击模型工具栏的按钮，生成代码的报告如图 8.5.32 所示。

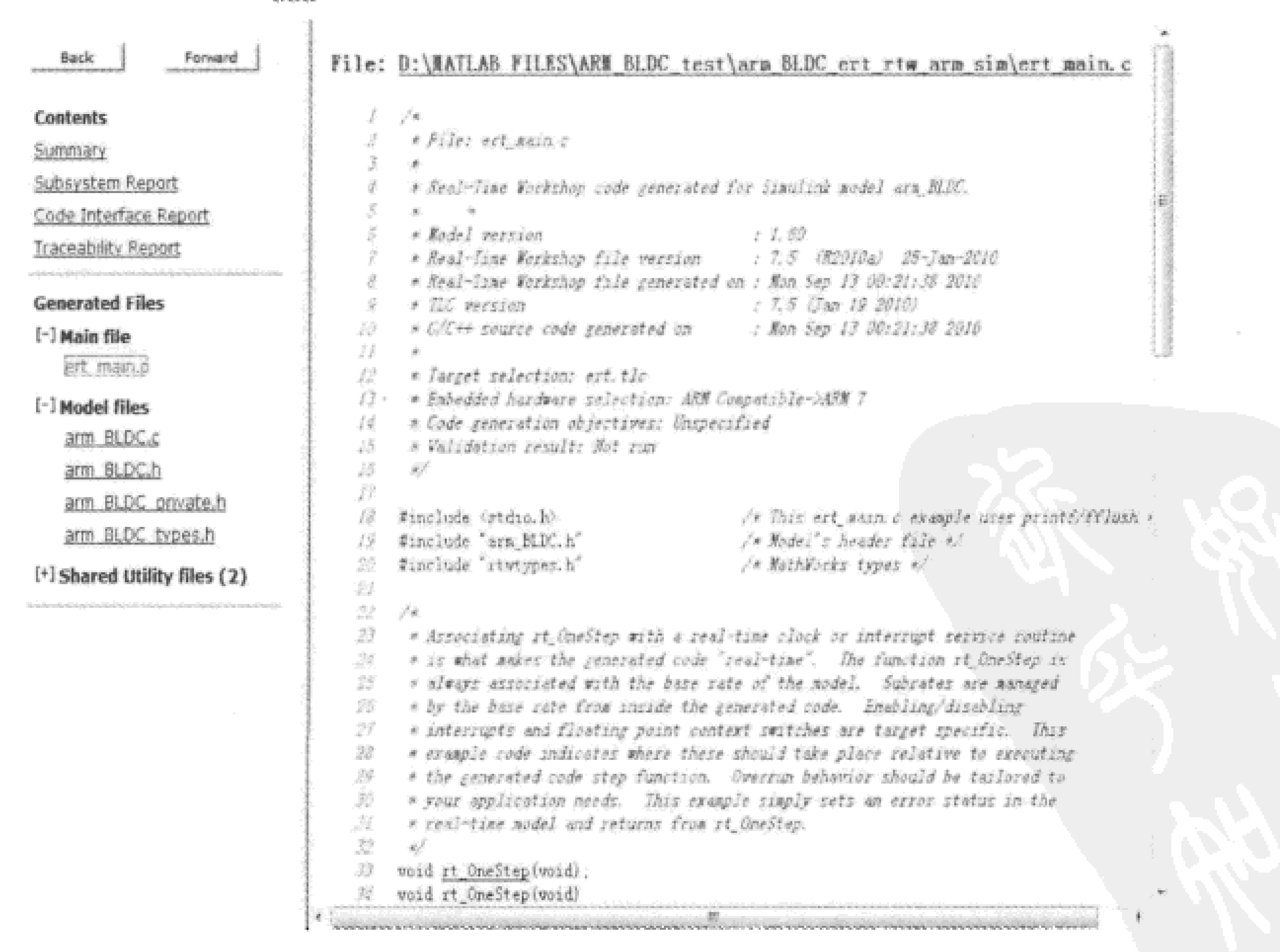


图 8.5.32　代码生成报告

系统自动打开 TASKING EDE,并加入生成的代码(图 8.5.33)。

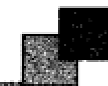


图 8.5.33 TASKING 工程

打开 ert_main.c 文件,做如下修改:

```
//#include <stdio.h>        /* 在后面的代码中删除了函数 printf/fflush,因此不需要 stdio.h */
#include "arm_BLDC.h"                                  /* 模型头文件 */
#include "rtwtypes.h"                                  /* MathWorks 数据类型 */
#include "LPC2124.h"                                   /* 添加 LPC2124 头文件 */
extern void PWM_Init(void);                            /* 声明外部函数 PWM_Init */
extern void AD_Read(void);                             /* 声明外部函数 AD_Read */
/*
 * Associating rt_OneStep with a real-time clock or interrupt service routine
 * is what makes the generated code "real-time". The function rt_OneStep is
 * always associated with the base rate of the model. Subrates are managed
 * by the base rate from inside the generated code. Enabling/disabling
 * interrupts and floating point context switches are target specific. This
 * example code indicates where these should take place relative to executing
 * the generated code step function. Overrun behavior should be tailored to
 * your application needs. This example simply sets an error status in the
 * real-time model and returns from rt_OneStep.
 */
void rt_OneStep(void);
void rt_OneStep(void)
{
  static boolean_T OverrunFlag = 0;

  /* Disable interrupts here */
```



```
    /* Check for overrun */
    if (OverrunFlag) {
  rtmSetErrorStatus(arm_BLDC_M, "Overrun");
      return;
    }

  OverrunFlag = TRUE;

    /* Save FPU context here (if necessary) */
    /* Re-enable timer or interrupt here */
    /* Set model inputs here */
      arm_BLDC_U.In1 = (IO0PIN&0x03800000)>>23;      //模型输入口与硬件相关联
      arm_BLDC_U.In2 = (IO0PIN&0x8000)>>15;
    /* Step the model */
  arm_BLDC_step();
    /* Get model outputs here */
        PINSEL0 = arm_BLDC_Y.Out1;                   //模型输出口与硬件相关联
        PINSEL1 = arm_BLDC_Y.Out2;
        IO0DIR = arm_BLDC_Y.Out3;
        PWMMR0 = arm_BLDC_Y.Out4;
        PWMMR1 = arm_BLDC_Y.Out5;
        PWMMR2 = arm_BLDC_Y.Out6;
        PWMMR3 = arm_BLDC_Y.Out7;
        PWMMR4 = arm_BLDC_Y.Out8;
        PWMMR5 = arm_BLDC_Y.Out9;
        PWMMR6 = arm_BLDC_Y.Out10;
        PWMLER = arm_BLDC_Y.Out11;
    /* Indicate task complete */
  OverrunFlag = FALSE;

    /* Disable interrupts here */
    /* Restore FPU context here (if necessary) */
    /* Enable interrupts here */
  }

  /*
   * The example "main" function illustrates what is required by your
   * application code to initialize, execute, and terminate the generated code.
   * Attaching rt_OneStep to a real-time clock is target specific. This example
   * illustates how you do this relative to initializing the model.
   */
  int_T main(int_T argc, const char_T *argv[]);
```

```
int_T main(int_T argc, const char_T *argv[])
{
  /* Initialize model */
arm_BLDC_initialize();
  PWM_Init();                                      //调用 PWM_Iint()
  ADCR = 0x002E0401;                               //ADC 初始化
  /* Simulating step behavior */
  while (rtmGetErrorStatus(arm_BLDC_M) == (NULL)) {
        AD_Read();                                 //调用 AD_Read()
        arm_BLDC_U.In3 = (ADDR>>6) & 0x3FF;        // A/D 转换结果赋值到 In3 口
        rt_OneStep();                              //调用 rt_OneStep()
  }

  /* Disable rt_OneStep() here */

  /* Terminate model */
arm_BLDC_terminate();
  return 0;
}

/*
 * File trailer for Real-Time Workshop generated code.
 *
 * [EOF]
 */
```

单击 EDE 环境的工具栏按钮 ，打开工程选项窗口(图 8.5.34)，在 Linker→Output Format 界面中，勾选 Intel Hex records for EPROM programmers(.hex)复选框。

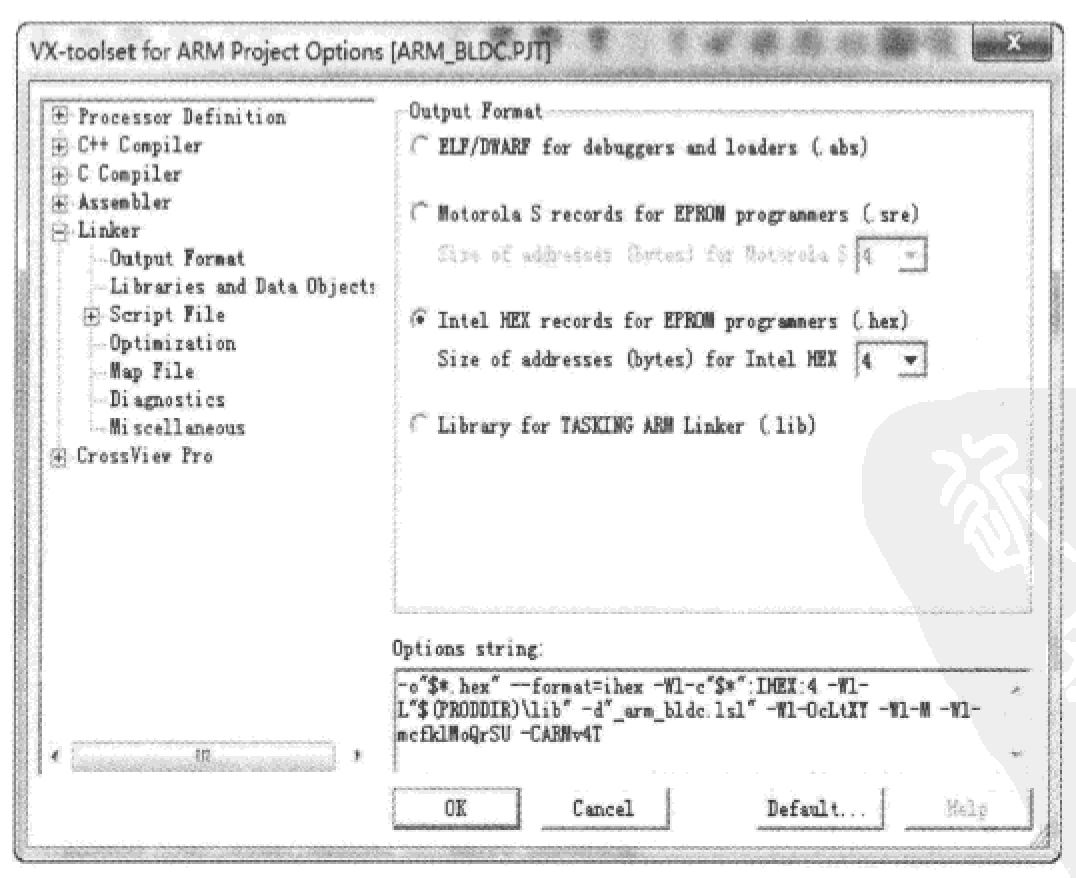
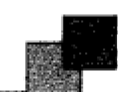


图 8.5.34　设置输出 HEX 文件



再单击工具栏按钮，编译工程，窗口下部的信息显示已成功生成.hex文件。

```
TASKING program builder v3.0r3 Build 088
Compiling "..\..\arm_bldc_ert_rtw_arm_sim\arm_bldc.c"
Assembling "arm_bldc.src"
Compiling "..\..\arm_bldc_ert_rtw_arm_sim\ert_main.c"
Assembling "ert_main.src"
Compiling "..\..\pwm_init.c"
Assembling "pwm_init.src"
Assembling "ad_read.src"
Linking and Locating to Intel Hex format files
```

值得一提的是 TASKING VX-toolset for ARM 开发环境针对 ARM 芯片做了特别的优化处理，编译后效率会优于 KEIL 开发环境。在 TASKING 工程所在目录下找到该文件，查看其属性，如图 8.5.35 所示，文件大小仅为 4KB 左右。

由于作者所使用的 Proteus 7.7 版本不支持 TASKING VX-toolset for ARM 生成的 HEX 文件格式，因此在后一小节的虚拟硬件测试中使用了由 KEIL 生成的.hex 文件。

若将 TSKING VX-toolset for ARM 生成的 hex 文件下载到合适的硬件执行，就不会受到此种情况的限制，但作者并未作此项测试，有兴趣的读者可自行测试。

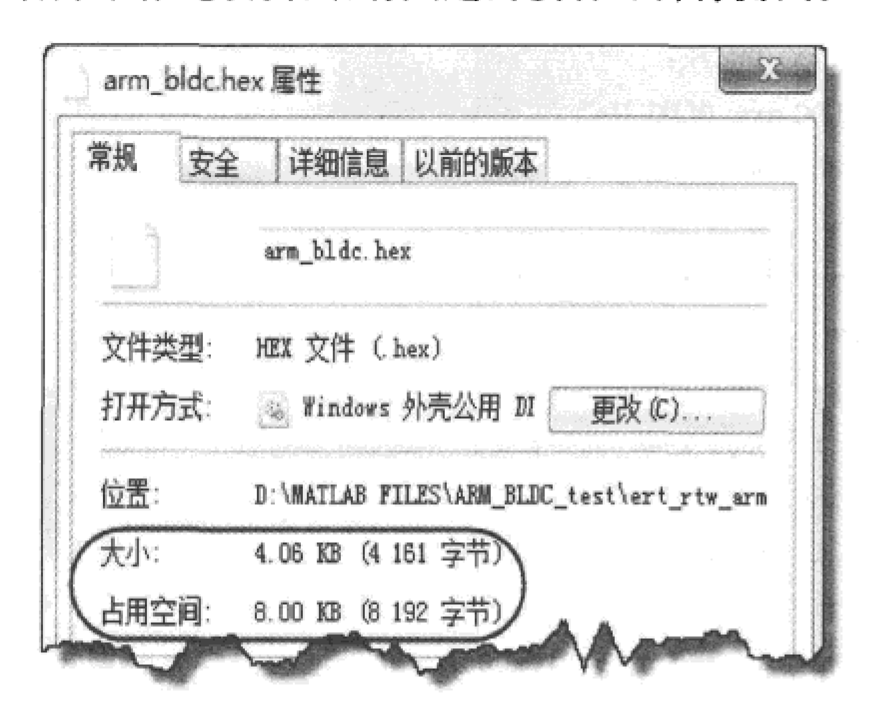


图 8.5.35 .hex 文件属性

8.5.8 代码效率比较

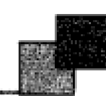
NXP 公司的官方网站上提供了基于 LPC2141 芯片的无刷电动机控制应用笔记，用户可以到网站 http://www.nxp.com/#/search/params=[q=AN10661,p=1,l=en]|filters=[]中下载文件 AN10661 Brushless DC motor control using the LPC2141，该应用笔记给出了完整代码(USB_Init/USB_connect 函数除外)。本例所建模型和 NXP 官方应用笔记中的实例功能基本相同，因此代码具有一定的可比性。

不同之处在于，本例的代码是基于 LPC2124 芯片的，而 NXP 采用的是 LPC2141 芯片。

这是由于将会在下一小节中进行 Proteus 虚拟硬件测试。遗憾的是 Proteus 并不支持 LPC2141 芯片，因此本例采用了相近的 LPC2124 代替。由此引起的代码差异是非常小的，基本可以忽略。

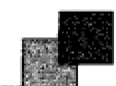
在自动生成的代码中，主要由 arm_BLDC.c 实现模型所表达的算法，其他文件为头文件、宏定义、函数声明等，有兴趣的读者可以自行查看。下面列出 arm_BLDC.c 代码(注释行已删除)：

```
#include"arm_BLDC.h"
#include"arm_BLDC_private.h"
#define arm_BLDC_IN_A                  (1U)
#definearm_BLDC_IN_B                   (2U)
#define arm_BLDC_IN_C                  (3U)
#define arm_BLDC_IN_D                  (4U)
#define arm_BLDC_IN_E                  (5U)
#define arm_BLDC_IN_F                  (6U)
#define arm_BLDC_IN_Init               (1U)
#define arm_BLDC_IN_MotorOff           (1U)
#define arm_BLDC_IN_MotorOn            (2U)
#define arm_BLDC_IN_NO_ACTIVE_CHILD    (0U)
BlockIO_arm_BLDC arm_BLDC_B;
D_Work_arm_BLDC arm_BLDC_DWork;
ExternalInputs_arm_BLDC arm_BLDC_U;
ExternalOutputs_arm_BLDC arm_BLDC_Y;
RT_MODEL_arm_BLDC arm_BLDC_M_;
RT_MODEL_arm_BLDC * arm_BLDC_M = &arm_BLDC_M_;
 void arm_BLDC_step(void)
{
  if (arm_BLDC_DWork.is_active_c1_arm_BLDC == 0) {
      arm_BLDC_DWork.is_active_c1_arm_BLDC = 1U;
      arm_BLDC_DWork.is_c1_arm_BLDC = arm_BLDC_IN_Init;
      arm_BLDC_B.io0dir = 0U;
      arm_BLDC_B.pinsel0 = (uint32_T)0x000A800A;
      arm_BLDC_B.pinsel1 = (uint32_T)0x00400800;
      arm_BLDC_DWork.is_Init = arm_BLDC_IN_MotorOff;
      arm_BLDC_B.pwmmr0 = 1024U;
      arm_BLDC_B.pwmmr1 = 0U;
      arm_BLDC_B.pwmmr2 = 0U;
      arm_BLDC_B.pwmmr3 = 0U;
      arm_BLDC_B.pwmmr4 = 0U;
      arm_BLDC_B.pwmmr5 = 0U;
      arm_BLDC_B.pwmmr6 = 0U;
      arm_BLDC_B.pwmler = (uint32_T)0x7F;
  } else {
     switch (arm_BLDC_DWork.is_Init) {
```



```
case arm_BLDC_IN_MotorOff:
 if (arm_BLDC_U.In2 == 1U) {
     arm_BLDC_DWork.is_Init = arm_BLDC_IN_MotorOn;
 }
 break;
case arm_BLDC_IN_MotorOn:
 if (arm_BLDC_U.In2 == 0U) {
     arm_BLDC_DWork.is_MotorOn = (uint8_T)arm_BLDC_IN_NO_ACTIVE_CHILD;
      arm_BLDC_DWork.is_Init = arm_BLDC_IN_MotorOff;
     arm_BLDC_B.pwmmr0  =  1024U;
     arm_BLDC_B.pwmmr1  =  0U;
     arm_BLDC_B.pwmmr2  =  0U;
     arm_BLDC_B.pwmmr3  =  0U;
     arm_BLDC_B.pwmmr4  =  0U;
     arm_BLDC_B.pwmmr5  =  0U;
     arm_BLDC_B.pwmmr6  =  0U;
     arm_BLDC_B.pwmler  =  (uint32_T)0x7F;
 } else if (arm_BLDC_U.In1 == 5U) {
    arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_A;
    arm_BLDC_B.pwmmr0  =  1024U;
    arm_BLDC_B.pwmmr1  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmmr2  =  0U;
    arm_BLDC_B.pwmmr3  =  0U;
    arm_BLDC_B.pwmmr4  =  0U;
    arm_BLDC_B.pwmmr5  =  0U;
    arm_BLDC_B.pwmmr6  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmler  =  (uint32_T)0x7F;
 } else if (arm_BLDC_U.In1 == 4U) {
    arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_B;
    arm_BLDC_B.pwmmr0  =  1024U;
    arm_BLDC_B.pwmmr1  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmmr2  =  0U;
    arm_BLDC_B.pwmmr3  =  0U;
    arm_BLDC_B.pwmmr4  =  0U;
    arm_BLDC_B.pwmmr5  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmmr6  =  0U;
    arm_BLDC_B.pwmler  =  (uint32_T)0x7F;
 } else if (arm_BLDC_U.In1 == 6U) {
    arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_C;
    arm_BLDC_B.pwmmr0  =  1024U;
    arm_BLDC_B.pwmmr1  =  0U;
    arm_BLDC_B.pwmmr2  =  0U;
    arm_BLDC_B.pwmmr3  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmmr4  =  0U;
    arm_BLDC_B.pwmmr5  =  arm_BLDC_U.In3;
    arm_BLDC_B.pwmmr6  =  0U;
    arm_BLDC_B.pwmler  =  (uint32_T)0x7F;
```

```
        } else if (arm_BLDC_U.In1 == 2U) {
          arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_D;
          arm_BLDC_B.pwmmr0 = 1024U;
          arm_BLDC_B.pwmmr1 = 0U;
          arm_BLDC_B.pwmmr2 = 0U;
          arm_BLDC_B.pwmmr3 = arm_BLDC_U.In3;
          arm_BLDC_B.pwmmr4 = arm_BLDC_U.In3;
          arm_BLDC_B.pwmmr5 = 0U;
          arm_BLDC_B.pwmmr6 = 0U;
          arm_BLDC_B.pwmler = (uint32_T)0x7F;
        } else if (arm_BLDC_U.In1 == 3U) {
          arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_E;
          arm_BLDC_B.pwmmr0 = 1024U;
          arm_BLDC_B.pwmmr1 = 0U;
          arm_BLDC_B.pwmmr2 = arm_BLDC_U.In3;
          arm_BLDC_B.pwmmr3 = 0U;
          arm_BLDC_B.pwmmr4 = arm_BLDC_U.In3;
          arm_BLDC_B.pwmmr5 = 0U;
          arm_BLDC_B.pwmmr6 = 0U;
          arm_BLDC_B.pwmler = (uint32_T)0x7F;
        } else {
          if (arm_BLDC_U.In1 == 1U) {
            arm_BLDC_DWork.is_MotorOn = arm_BLDC_IN_F;
            arm_BLDC_B.pwmmr0 = 1024U;
            arm_BLDC_B.pwmmr1 = 0U;
            arm_BLDC_B.pwmmr2 = arm_BLDC_U.In3;
            arm_BLDC_B.pwmmr3 = 0U;
            arm_BLDC_B.pwmmr4 = 0U;
            arm_BLDC_B.pwmmr5 = 0U;
            arm_BLDC_B.pwmmr6 = arm_BLDC_U.In3;
            arm_BLDC_B.pwmler = (uint32_T)0x7F;
          }
        }
        break;
      default:
        arm_BLDC_DWork.is_Init = arm_BLDC_IN_MotorOff;
        arm_BLDC_B.pwmmr0 = 1024U;
        arm_BLDC_B.pwmmr1 = 0U;
        arm_BLDC_B.pwmmr2 = 0U;
        arm_BLDC_B.pwmmr3 = 0U;
        arm_BLDC_B.pwmmr4 = 0U;
        arm_BLDC_B.pwmmr5 = 0U;
        arm_BLDC_B.pwmmr6 = 0U;
        arm_BLDC_B.pwmler = (uint32_T)0x7F;
        break;
      }
    }
```



```
  arm_BLDC_Y.Out1 = arm_BLDC_B.pinsel0;
  arm_BLDC_Y.Out2 = arm_BLDC_B.pinsel1;
  arm_BLDC_Y.Out3 = arm_BLDC_B.io0dir;
  arm_BLDC_Y.Out4 = arm_BLDC_B.pwmmr0;
  arm_BLDC_Y.Out5 = arm_BLDC_B.pwmmr1;
  arm_BLDC_Y.Out6 = arm_BLDC_B.pwmmr2;
  arm_BLDC_Y.Out7 = arm_BLDC_B.pwmmr3;
  arm_BLDC_Y.Out8 = arm_BLDC_B.pwmmr4;
  arm_BLDC_Y.Out9 = arm_BLDC_B.pwmmr5;
  arm_BLDC_Y.Out10 = arm_BLDC_B.pwmmr6;
  arm_BLDC_Y.Out11 = arm_BLDC_B.pwmler;
}
void arm_BLDC_initialize(void)
  rtmSetErrorStatus(arm_BLDC_M, (NULL));
  (void) memset(((void * ) &arm_BLDC_B), 0,
                sizeof(BlockIO_arm_BLDC));
  (void) memset((void * )&arm_BLDC_DWork, 0,
                sizeof(D_Work_arm_BLDC));
  (void) memset((void * )&arm_BLDC_U, 0,
                sizeof(ExternalInputs_arm_BLDC));
  (void) memset((void * )&arm_BLDC_Y, 0,
                sizeof(ExternalOutputs_arm_BLDC));
  arm_BLDC_DWork.is_Init = 0U;
  arm_BLDC_DWork.is_MotorOn = 0U;
  arm_BLDC_DWork.is_active_c1_arm_BLDC = 0U;
  arm_BLDC_DWork.is_c1_arm_BLDC = 0U;
  arm_BLDC_B.pinsel0 = 0U;
  arm_BLDC_B.pinsel1 = 0U;
  arm_BLDC_B.io0dir = 0U;
  arm_BLDC_B.pwmmr0 = 0U;
  arm_BLDC_B.pwmmr1 = 0U;
  arm_BLDC_B.pwmmr2 = 0U;
  arm_BLDC_B.pwmmr3 = 0U;
  arm_BLDC_B.pwmmr4 = 0U;
  arm_BLDC_B.pwmmr5 = 0U;
  arm_BLDC_B.pwmmr6 = 0U;
  arm_BLDC_B.pwmler = 0U;
}
void arm_BLDC_terminate(void)
{
  }
```

NXP 公司提供的代码中 adc.c、bldc.c、hsensor.c、PWM.c、timer1.c 代码均为实现算法所必须，bldc.c 为函数声明文件读者可自行查看。下面列出其算法代码。

Adc.c 文件

```
#include <LPC214x.h>
  void ADC0_Init(void)
```

```
{
PINSEL1 |= 0x00040000; // P0.25 = AIN0.4
AD0CR = 0x00200F10; // initialise ADC0, select AIN4
AD0CR |= 0x00010000; // start burst mode now, see errata ADC.2
}
```

Bldc.c 文件

```
#include <LPC214x.H> // LPC214x definitions
#include "bldc.h"

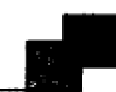
unsigned char actualSpeed = 0;
unsigned char desiredSpeed = 0;
unsigned int RPM,fRPM;

void GetInReport(unsigned char * rep) // Host is asking for an InReport
{
rep[0] = fRPM; // send measured motor speed (low byte)
rep[1] = fRPM >> 8; // send measured motor speed (high byte)
rep[2] = AD0DR4 >> 8;; // send potm value for debugging
}

void SetOutReport(unsigned char * rep) // OutReport received from USB host
{
if (rep[0] < 101)
desiredSpeed = rep[0]; // New desired speed value received
}

int main (void)
{
ADC0_Init(); // ADC0 Initialization
T1_Init(); // 10 msec tick
PWM_Init(); // PWM Timer Initialization
HES_Init();
// USB_Init(); // USB Initialization
// USB_Connect(1); // USB Connect

while (1) // Loop forever
{
if (((AD0DR4 >> 8) & 0xFF) > MAX_Im) // Check motor overcurrent
{
VICIntEnClr = 0xFFFFFFFF; // disable all interrupts!
PWMMR1 = 0; // Q1 off
PWMMR2 = 0; // Q2 off
PWMMR3 = 0; // Q3 off
PWMMR4 = 0; // Q4 off
PWMMR5 = 0; // Q5 off
PWMMR6 = 0; // Q6 off
```



```
PWMLER = 0x7F; // enable PWM0-PWM6 match latch (reload)
while (1) ; // wait for a RESET
}
if (f_10ms) // every 10 mseconds
{
f_10ms = 0;

if (actualSpeed > desiredSpeed)
actualSpeed -- ;
else if (actualSpeed < desiredSpeed)
actualSpeed ++ ;

RPM = 10000000 / T0CR0; // calculate motor speed
fRPM = ((fRPM * 15) + RPM) / 16; // filter it
}
}
}
```

Hsensor. c 文件

```
#include <LPC214x.H> // LPC214x definitions
#include "bldc.h"

__irq void T0_Isr(void)
{
T0TC = 0; // Reset timer

switch ((IO0PIN >> 18) & 7) // read Hall sensor inputs P0.18, P0.19 and P0.20
{
case 1: PWMMR1 = actualSpeed; // phase 6: 001
PWMMR2 = 0;
PWMMR3 = 0;
PWMMR4 = 0;
PWMMR5 = 0;
PWMMR6 = actualSpeed;
break;
case 2: PWMMR1 = 0; // phase 4: 010
PWMMR2 = actualSpeed;
PWMMR3 = 0;
PWMMR4 = actualSpeed;
PWMMR5 = 0;
PWMMR6 = 0;
break;
case 3: PWMMR1 = 0; // phase 5: 011
PWMMR2 = actualSpeed;
PWMMR3 = 0;
PWMMR4 = 0;
PWMMR5 = 0;
```

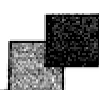
```
PWMMR6 = actualSpeed;
break;
case 4: PWMMR1 = 0; // phase 2: 100
PWMMR2 = 0;
PWMMR3 = actualSpeed;
PWMMR4 = 0;
PWMMR5 = actualSpeed;
PWMMR6 = 0;
break;
case 5: PWMMR1 = actualSpeed; // phase 1: 101
PWMMR2 = 0;
PWMMR3 = 0;
PWMMR4 = 0;
PWMMR5 = actualSpeed;
PWMMR6 = 0;
break;
case 6: PWMMR1 = 0; // phase 3: 110
PWMMR2 = 0;
PWMMR3 = actualSpeed;
PWMMR4 = actualSpeed;
PWMMR5 = 0;
PWMMR6 = 0;
break;
default: break; // invalid
}

T0IR = 0xFF; // reset flags
PWMLER = 0x7F; // enable PWM0 - PWM6 match latch (reload)
VICVectAddr = 0; // Acknowledge interrupt by reseting VIC
}

void HES_Init(void)
{
VICVectAddr1 = (unsigned int) &T0_Isr;
VICVectCntl1 = 0x24; // Channel1 on Source#4 ... enabled
VICIntEnable |= 0x10; // Channel#4 is the Timer 0

PINSEL1 |= 0x3A000000; // P0.30,P0.28,P0.29 as CAP0.0,CAP0.2,CAP0.3

T0PR = 60; // pre 60, timer runs at 60 MHz / 60 = 1 MHz
T0MR0 = 1000000; // = 1 sec / 1 us
T0MCR = 3;
T0CCR = 0x0FC7; // Capture on both edges and enable the interrupt
T0TC = 0; // Reset timer
T0TCR = 1; // start timer
}
```



PWM. c 文件

```
# include < LPC214x. h>

void PWM_Init(void)
{
PINSEL0 | = 0x000A800A; // select PWM1-4 and PWM6
PINSEL1 | = 0x00000400; // select PWM5

PWMPR = 20; // prescaler to 20, timer runs at 60 MHz / 20 = 3 MHz
PWMPC = 0; // prescale counter to 0
PWMTC = 0; // reset timer to 0
PWMMR0 = 100; // -> PMW base frequency = 3 MHz / 100 = 30 KHz
PWMMR1 = 0; // Match 1 for Q1 (off)
PWMMR2 = 0; // Match 2 for Q2 (off)
PWMMR3 = 0; // Match 3 for Q3 (off)
PWMMR4 = 0; // Match 4 for Q4 (off)
PWMMR5 = 0; // Match 5 for Q5 (off)
PWMMR6 = 0; // Match 6 for Q6 (off)
PWMMCR = 0x00000002; // reset TC on MR0
PWMPCR = 0x7E00; // enable PWM1 - PWM6 outputs
PWMLER = 0x7F; // enable PWM0 - PWM6 match latch (reload)
PWMTCR = 0x09; // enable PWM mode and start timer
}
```

Timer1. c 文件

```
# include < LPC214x. H> // LPC214x definitions

char f_10ms = 0;

__irq void T1_Isr(void) // Timer 1 ISR every 10 msec
{
f_10ms = 1; // toggles every 10 mseconds
T1IR = 0x01; // reset interrupt flag
VICVectAddr = 0; // reset VIC
}

void T1_Init(void)
{
VICVectAddr2 = (unsigned int) &T1_Isr;
VICVectCntl2 = 0x25; // Channel2 on Source # 5 ... enabled
VICIntEnable | = 0x20; // Channel # 5 is the Timer 1

T1MR0 = 600000; // = 10 msec / 16,67 nsec
T1MCR = 3; // Interrupt on Match0, reset timer on match
// Pclk = 60 MHz, timer count = 16,67 nsec
T1TC = 0; // reset Timer counter
```

```
T1TCR = 1; // enable Timer
}
```

通过代码量的比较，可以较直观地看出，自动生成的代码已经很接近手写代码的效率了。下面将进一步比较代码编译后的.hex 文件。

在 Keil 中建立工程，将修改过的自动生成代码添加到工程中，编译生成.hex 文件，如图 8.5.36 所示。

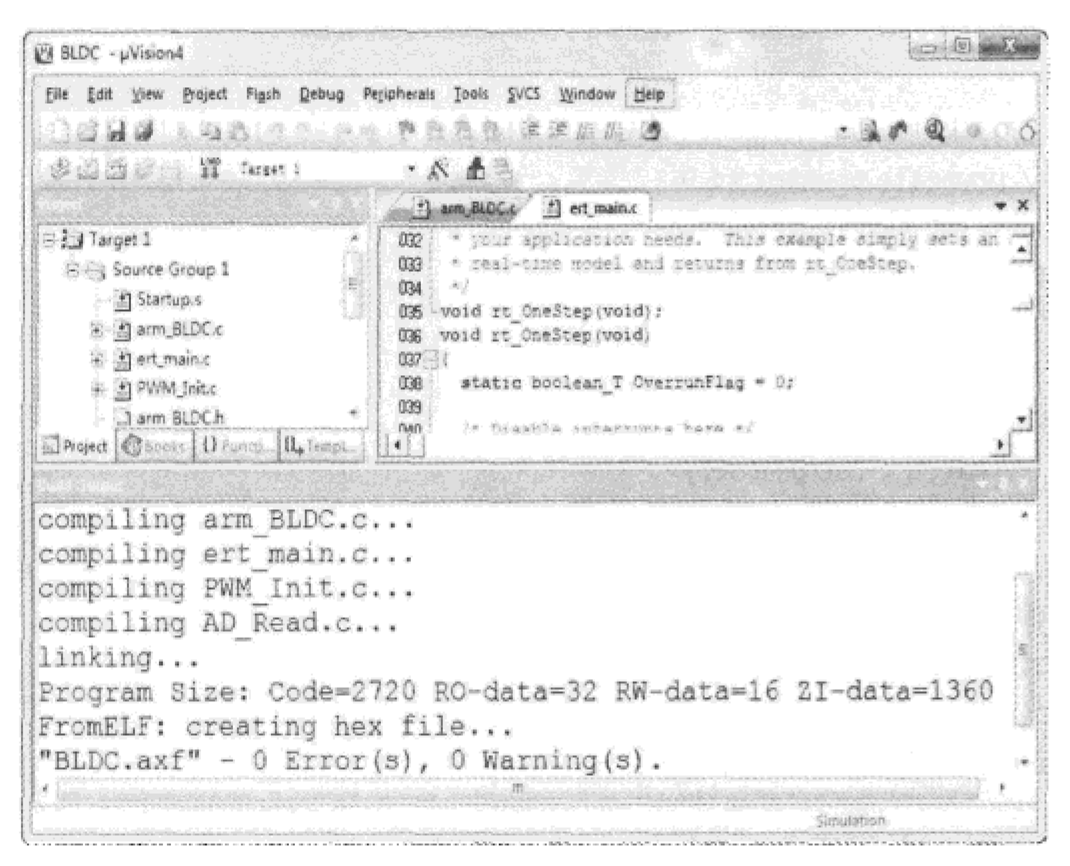


图 8.5.36　编译信息

在工程目录下可以找到生成的.hex 文件，查看其属性。可以看到文件大小为 7.65KB，如图 8.5.37 所示。

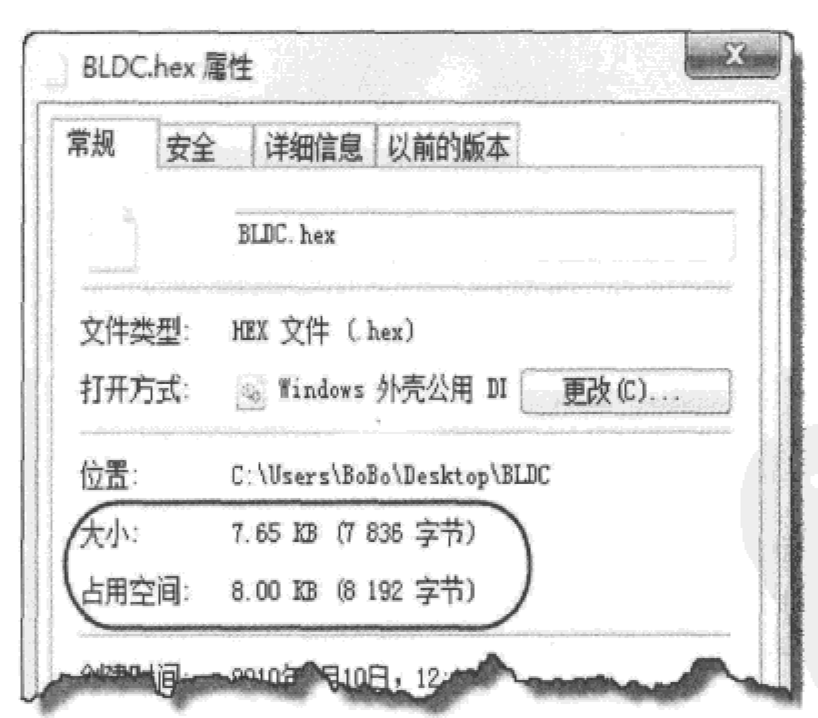
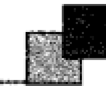


图 8.5.37　HEX 文件属性

接下来，建立一个新的 Keil 工程，加入 NXP 官方代码，编译生成.hex 文件，如图 8.5.38 所示。



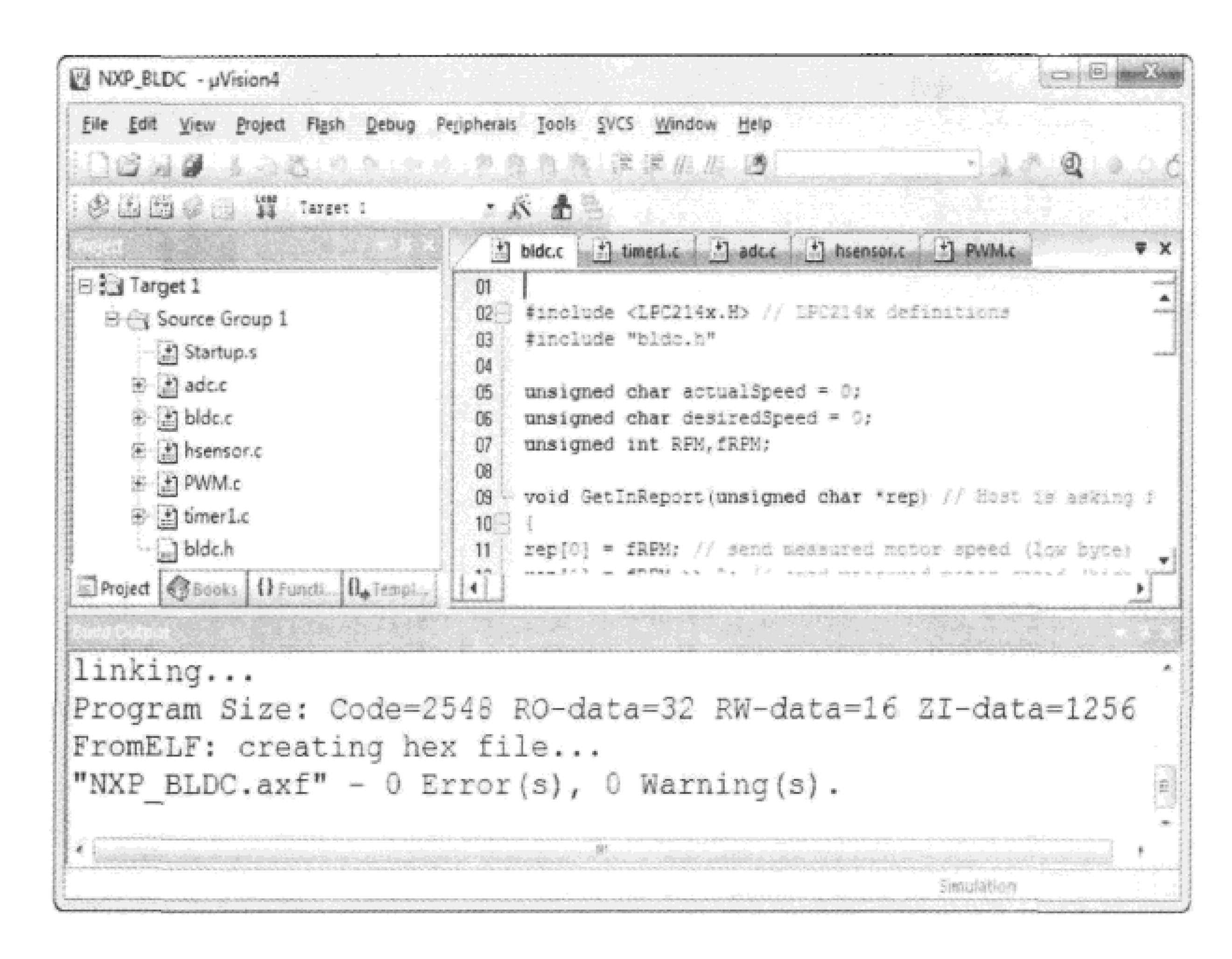


图 8.5.38 编译信息

在工程目录下找到生成的.hex文件，查看其属性。可以看到文件大小为7.18KB，如图8.5.39所示。这再次证明了自动生成代码的效率并不逊色于手写代码。

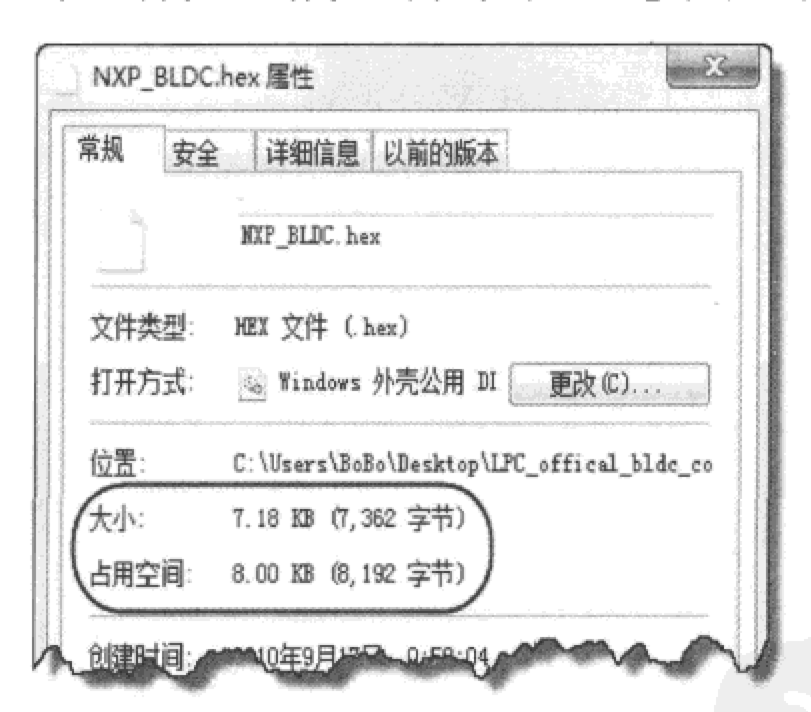


图 8.5.39 .hex文件属性

8.5.9 虚拟硬件测试

建立驱动无刷电动机模型。桥式逆变电路控制电动机驱动电动机正常运转；IR2112提供足够的导通电压；Logicstate控制电动机是否开始工作；电位器调节输入电压，控制电动机转速；示波器显示6路PWM信号的波形，如图8.5.40所示。

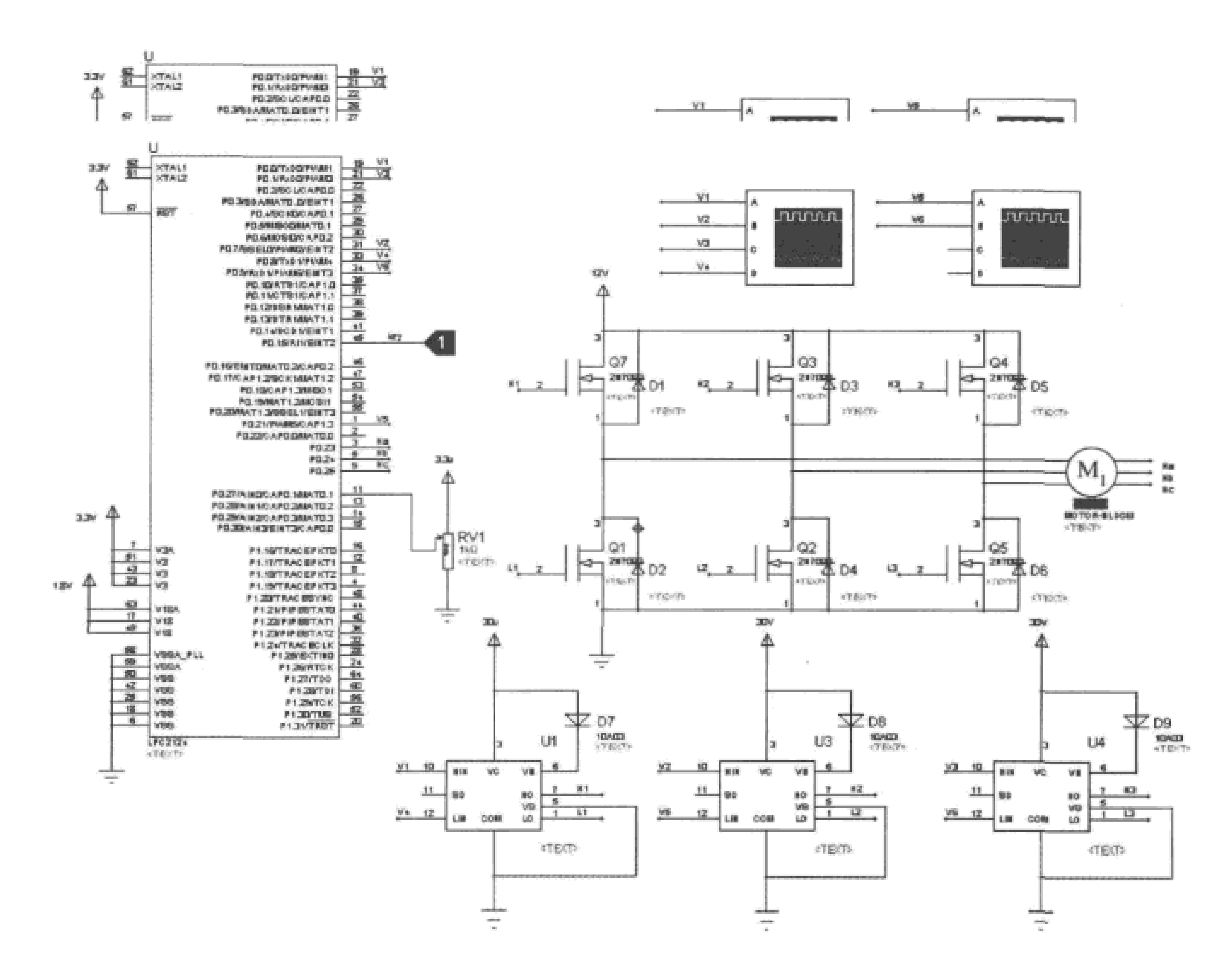


图 8.5.40 Proteus 原理图

由于使用元件较多，为避免混乱，采用网络标号的方法连接元件。分别在在 LPC2124 芯片的 P0.0、P0.1、P0.7、P0.8、P0.9、P0.21 引脚标注 V1、V3、V2、V4、V6、V5，输出 6 路 PWM 信号；P0.23、P0.24、P0.25 引脚标注 Ha、Hb、Hc，表示霍尔元件的输入；P0.15 直接连接 Logicstate，P0.27 直接连接电位器，如图 8.5.41 所示。

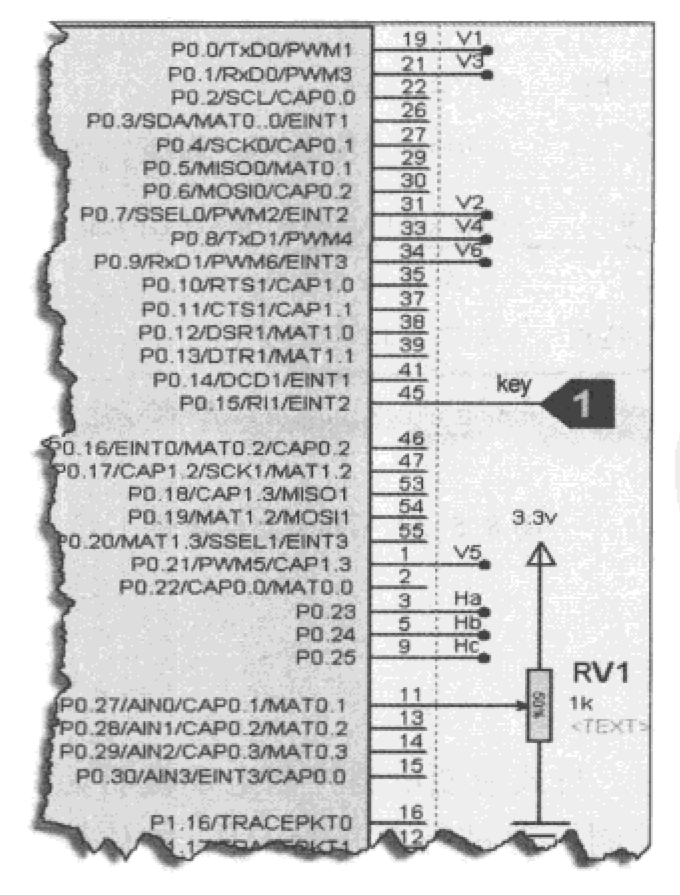
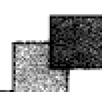


图 8.5.41 元件引脚连接



IR2112 芯片按图 8.5.42 连接，三个芯片的 HIN 引脚分别标注 V1～V3，LIN 引脚标注 V4～V6；HO 引脚标注 H1～H3，LO 引脚标注 L1～L3。

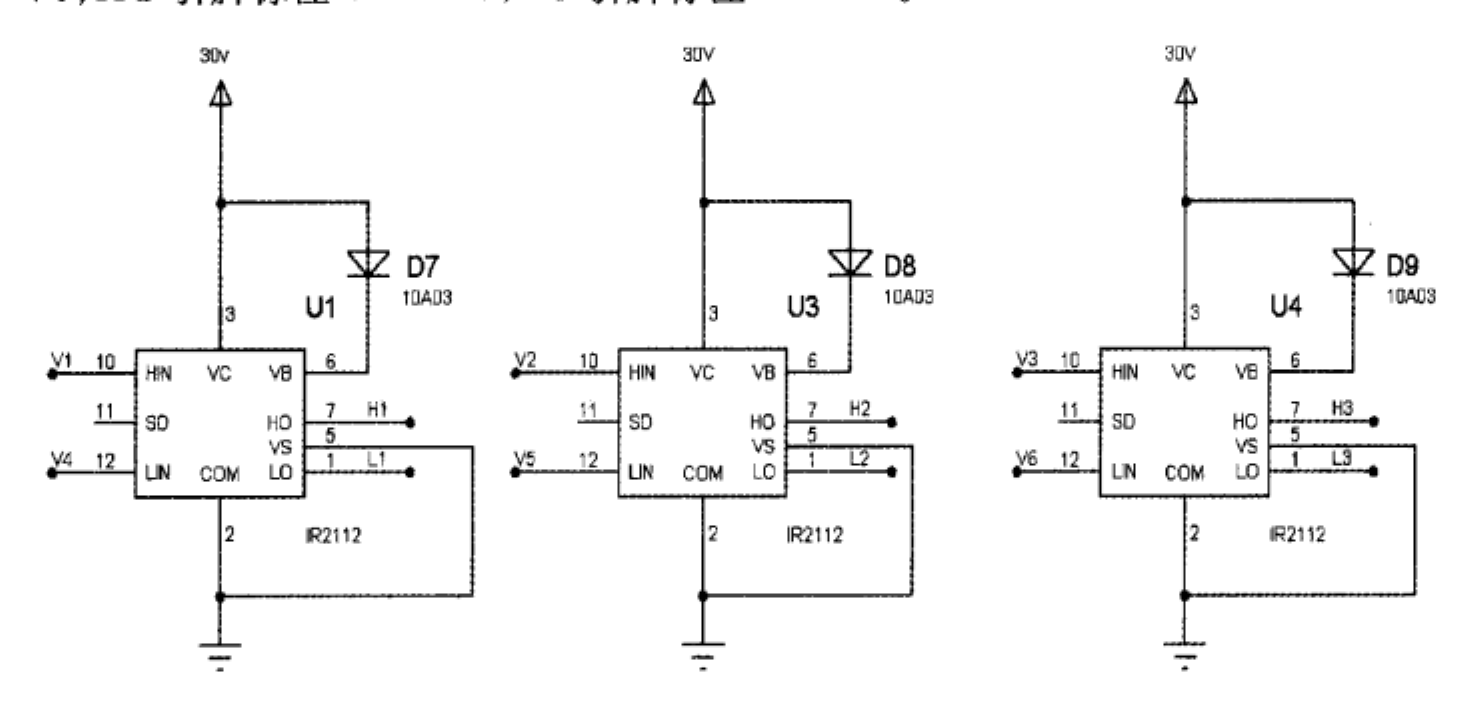


图 8.5.42　IR2112 引脚连接

在桥式逆变器的功率管引脚处标注 H1～H3，L1～L3，与 IR2112 的对应引脚相连接；电动机右侧的三根引脚分别标注 Ha、Hb、Hc，将霍尔传感器的值输入到芯片中去，如图 8.5.43 所示。

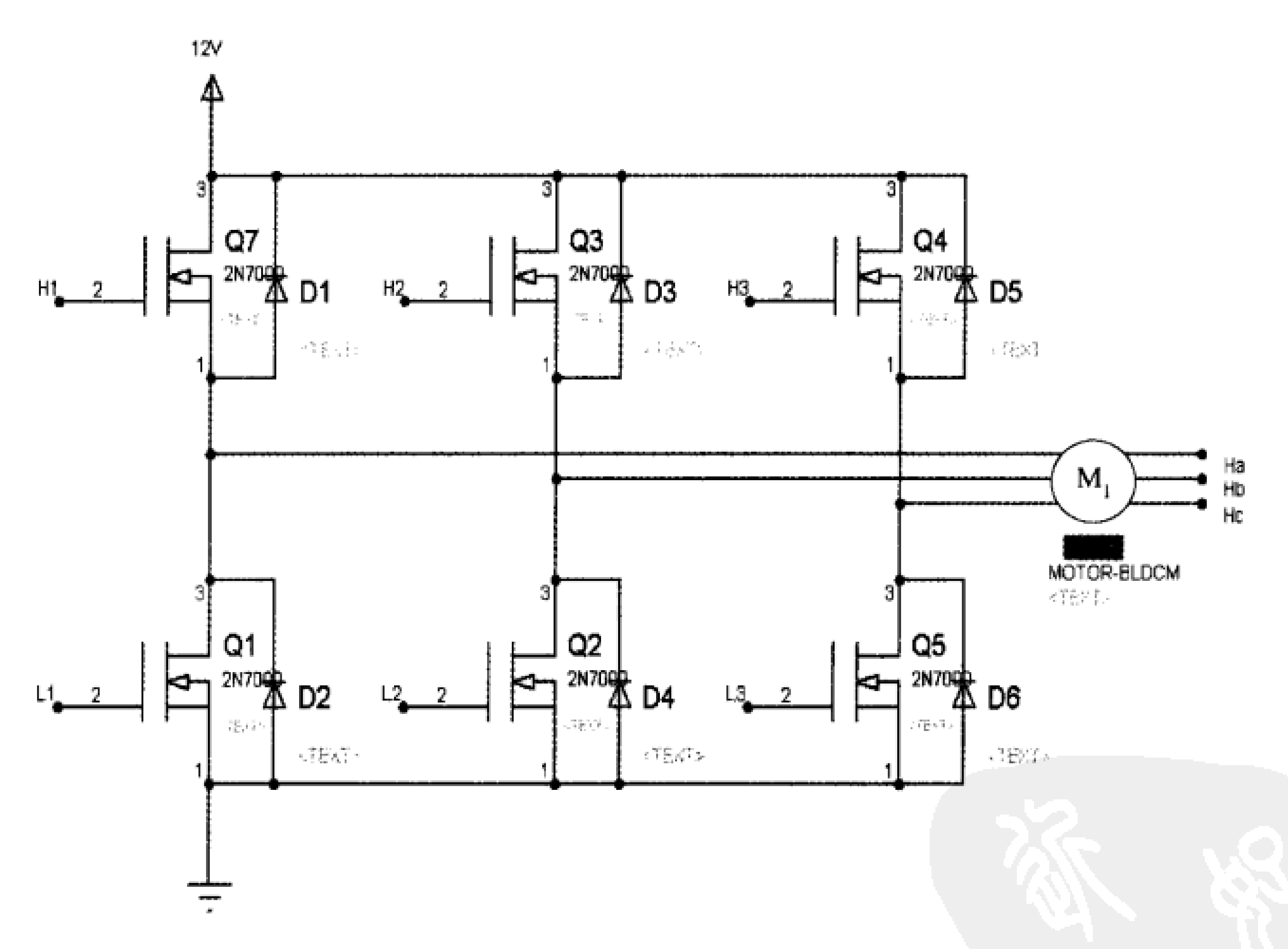


图 8.5.43　元件设置

选择左侧示波器中的 4 根引脚，标注 V1～V4，右侧示波器的 A，B 两根引脚，标注 V5，V6，用以显示 6 路 PWM 输出信号。

由于 IR2112 芯片的输入电压（HIN、LIN）不能低于 V_{DD}-0.3 V，而 LPC2124 输出的 PWM 波为 3.3 V，proteus 系统默认的 V_{DD}为 5 V，这会导致 IR2112 无法正常工作。在菜单栏上选择 Design→Config Power Rails，将 V_{DD}设置为 3.3 V，如图 8.5.44 所示。

完成上述设置后，加载之前由 Keil 生成的.hex 文件，单击“仿真”按钮。可以看到，电动机正常运行，符合模型预期功能，如图 8.5.45 所示。

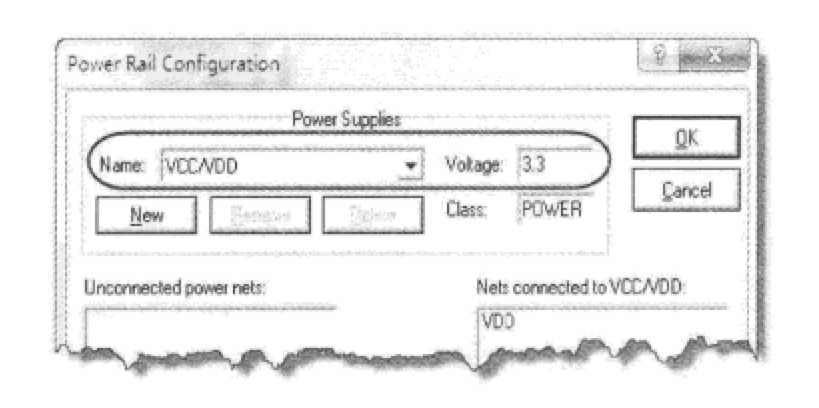


图 8.5.44　设置 V_{DD}

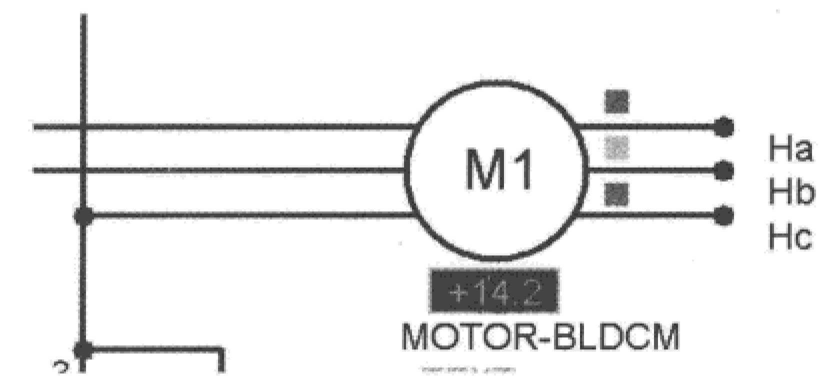


图 8.5.45　仿真结果

由图 8.5.45 可知，当前时刻由霍尔传感器输出的信号为 010，对应的十进制数为 2，当 sensor 值为 2 时，PWM 输出波形如图 8.5.46、图 8.5.47 所示。

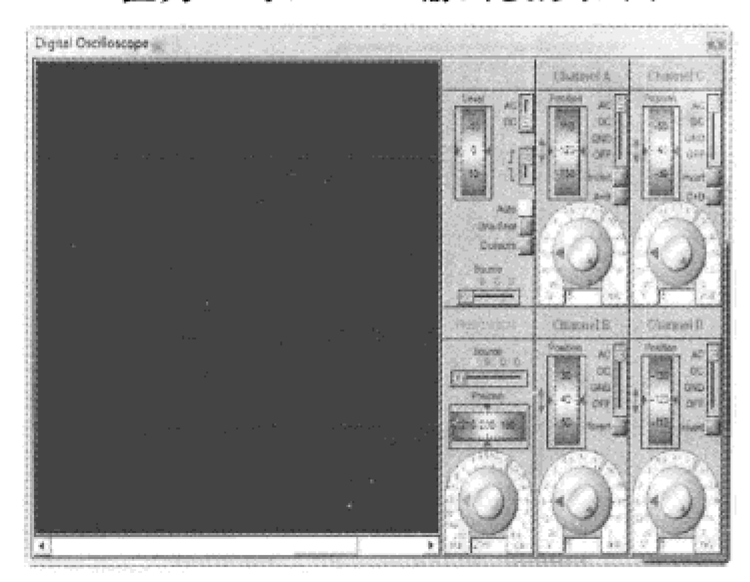


图 8.5.46　输出 PWM 波形

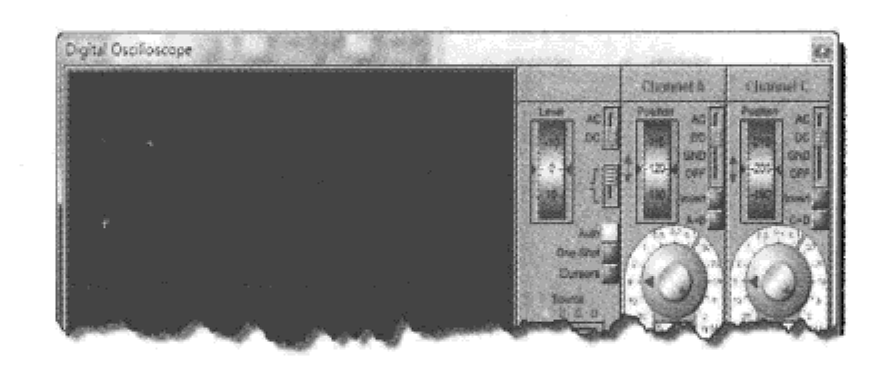


图 8.5.47　输出 PWM 波形

用户还可以再添加一个示波器，将示波器的 A、B、C 三根引脚连接到电动机左侧的三个引脚，可以观察到图 8.5.48 所示的波形。在一个完整的周期内，三路信号呈 120°相位差。

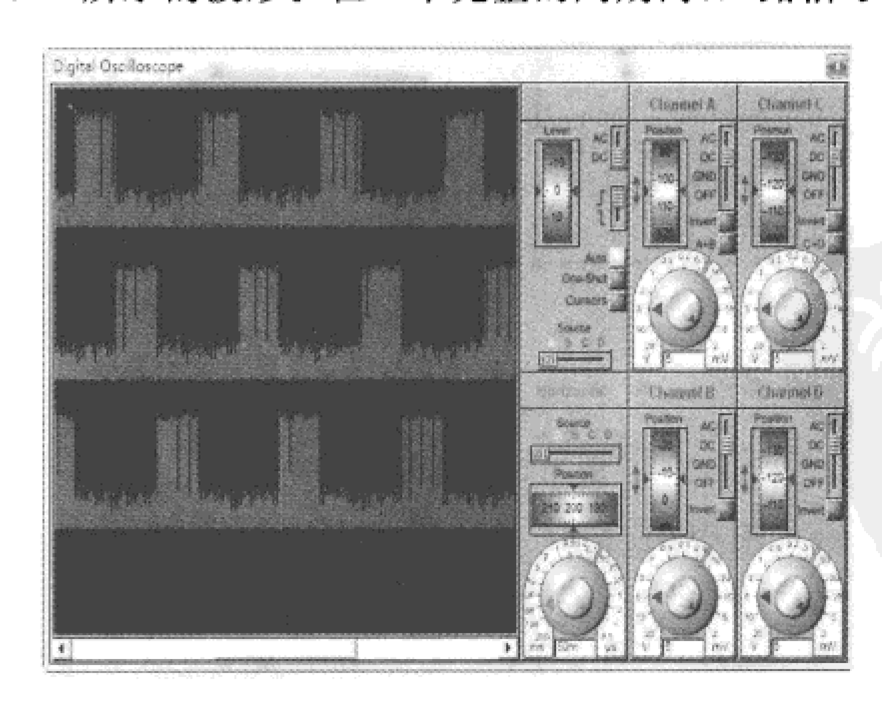


图 8.5.48　输入电动机的波形

第 9 章 基于模型的设计

本章以第 2 章电动机的 PID 控制模型为例，介绍基于模型的设计的完整过程，它是对本书的总结，也是本书的核心内容。

在本章中，作者给出了符合 DO-178b 航空电子规范的基于模型设计的工作流程。并按照这个设计流程，对电动机的 PID 控制模型进行了相关测试与验证。由于作者的 MATLAB 插件中缺少 Polyspace 插件(Polyspace 插件售价据说在 10 万美元左右)，无法对模型生成的代码进行运行时的错误检查。但是作为一个完整的基于模型的设计，对生成的代码进行运行时的错误检查是必不可少的，特别是对于某些对可靠性及稳定性要求非常高的项目，例如航天与军工行业，运行时错误检查是极为重要的。否则 20 世纪 80 年代，欧洲阿丽亚娜火箭发射失败的悲剧可能重演，带来的将是灾难性的后果。

考虑到介绍基于模型设计的完整流程会占用大量篇幅，作者对第 5 章到第 8 章的基于模型的设计的工作流程做了简化。因此前面几章并不是真正意义上的基于模型的设计，只能算是 MCU 器件嵌入式 C 代码的快速生成。

如果读者想得到高效的、高可靠性的嵌入式实时 C 代码，基于模型设计的工作流程必须参照本章介绍的步骤进行。

本章的主要内容：

- 传统设计的弊端。
- 基于模型设计的优势。
- 基于模型设计的流程。
- 需求分析及跟踪。
- 模型检查及验证。
- 定点模型。
- 软件在环测试。
- 代码跟踪。
- 代码优化及代码生成。
- 虚拟硬件测试。



9.1 传统设计的弊端

传统设计的工作流程如图 9.1.1 所示。

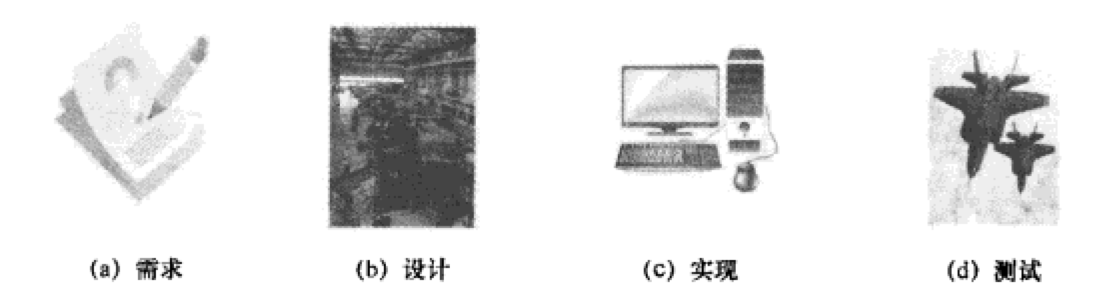


图 9.1.1 传统设计的工作流程

传统设计分需求→设计→实现→测试 4 个阶段，它的主要缺陷如下：

(1) 这 4 个阶段彼此孤立，重复劳动严重。

(2) 工程师们不可避免地存在对需求分析与技术规范文档的理解差异，埋下失败的伏笔。

(3) 在设计阶段需要打造硬件平台(不能保证满足技术规范的指标)，前期资金投入大。

(4) 在实现阶段只能采用手工编程的方式，人员素质要求高、难度大、效率低、错误多。

(5) 测试阶段只能在完成原型样机之后才能进行，查错与修正的费用巨大，造成潜在的市场风险。

据相对资料描述，超过 50%的错误是在编制技术规范阶段引入的，而这时能够发现的错误仅有 8%，大部分的错误需要等到测试阶段才能发现，如图 9.1.2、图 9.1.3 所示。

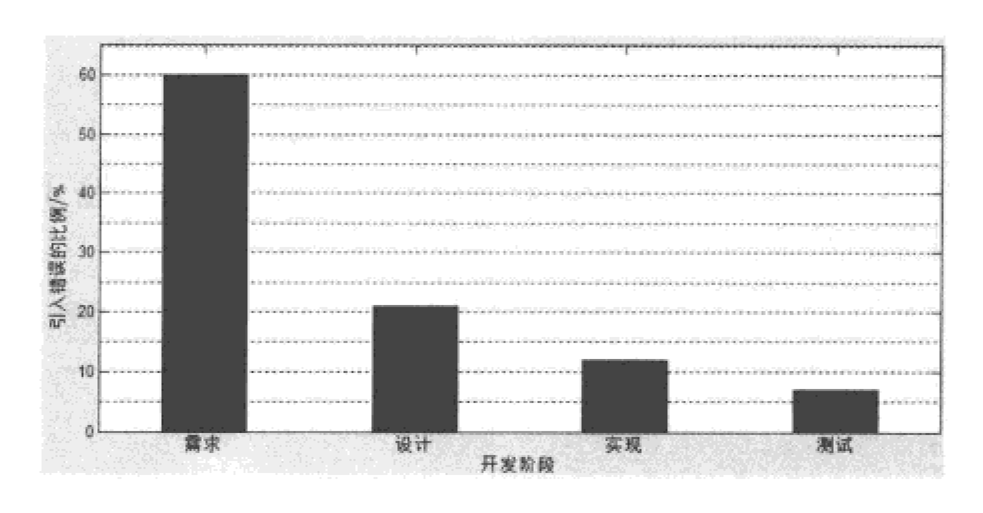


图 9.1.2 各开发阶段引入错误的比例

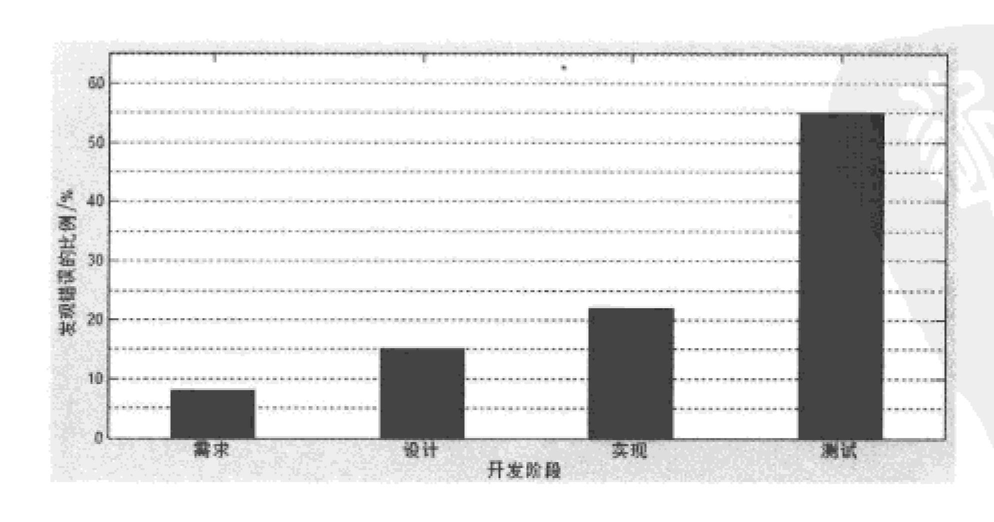


图 9.1.3 各开发阶段发现错误的比例



(6) 对于大型项目，参加人员多，开发平台不统一，给后期整合带来麻烦。

9.2 基于模型设计的优势

相对于传统的设计模式，基于模型设计的4个阶段相互联系（图9.1.1），它的优势如下：

(1) 在统一的开发-测试平台上，让设计从需求分析阶段就开始验证与确认，并做到持续不断地验证与测试，让设计的缺陷暴露在开发的初级阶段。



图 9.2.1 基于模型设计的工作流程

(2) 让工程师把主要精力放在算法和测试用例的研究上，嵌入式C代码的生成与验证留给计算机去自动完成，自动生成C代码。

据相关资料介绍，自动生成代码在某些情况下已能够到达甚至超过手写代码的效率，如表9.2.1和表9.2.2所列。本书第8章的无刷电动机控制（模型未经优化）的例子，模型自动生成的C代码已非常接近NXP提供的手写C代码的水平，编译后生成的HEX文件大小也相差不大（图9.2.2、图9.2.3）。

表 9.2.1 美国伟世通(Visteon)公司数据

类型	手写代码	自动代码生成	类型	手写代码	自动代码生成
ROM	6408	6192	RAM	132	112

表 9.2.2 美国GM(General Motors)公司数据

类型	手写代码	自动代码生成	类型	手写代码	自动代码生成
Calibration ROM	9464	9464	ROM	2952	2900
RAM	240	238			

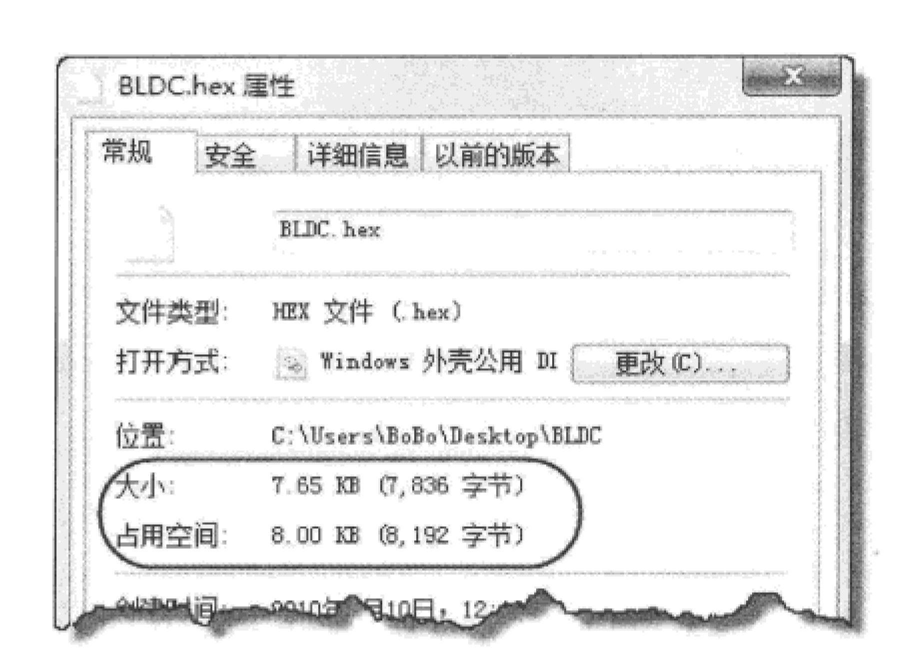


图 9.2.2　自动生成代码经 Keil 编译的 HEX 文件

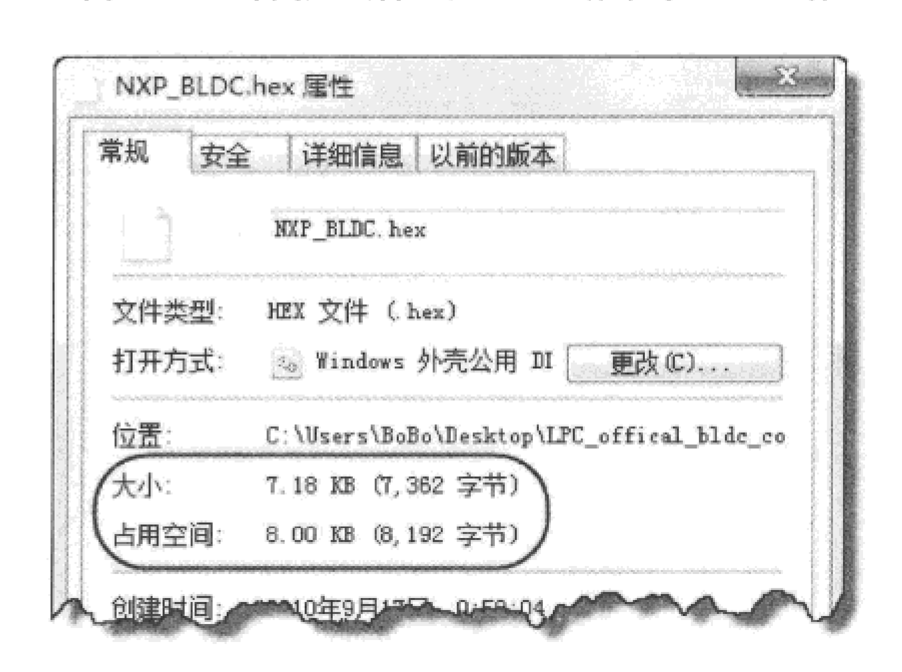


图 9.2.3　NXP 官方代码经 Keil 编译的 HEX 文件

(3) 大大缩短开发周期与降低开发成本。Arthur D. Little 公司的调研项目显示,使用基于模型的设计,其开发成本大大低于传统的开发方式。随着时光的流逝,基于模型设计的优势将更加明显,如图 9.2.4 所示。

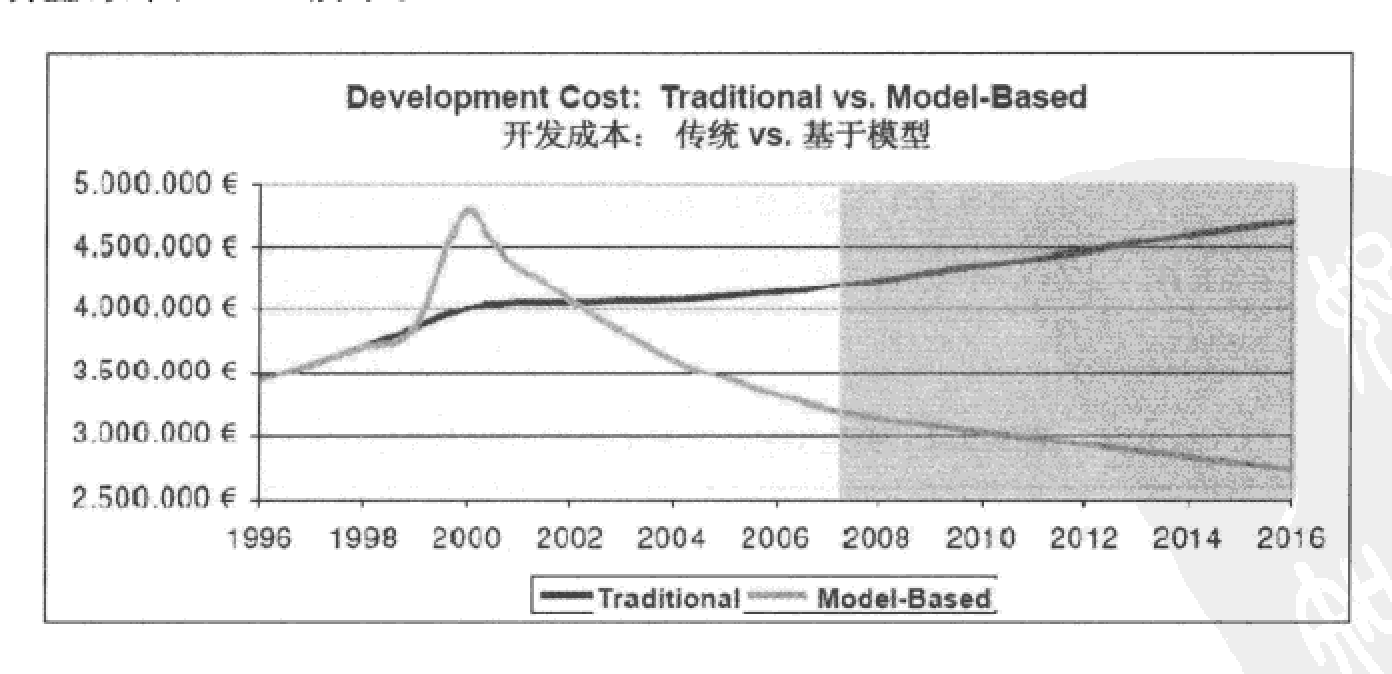
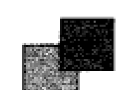


图 9.2.4　开发成本比较



9.3 基于模型设计的流程

满足 DO-178b 航空电子规范的基于模型设计的工作流程如图 9.3.1 所示。用户可根据自己对生成代码安全性的要求，对图 9.3.1 所示的工作流程进行剪裁。

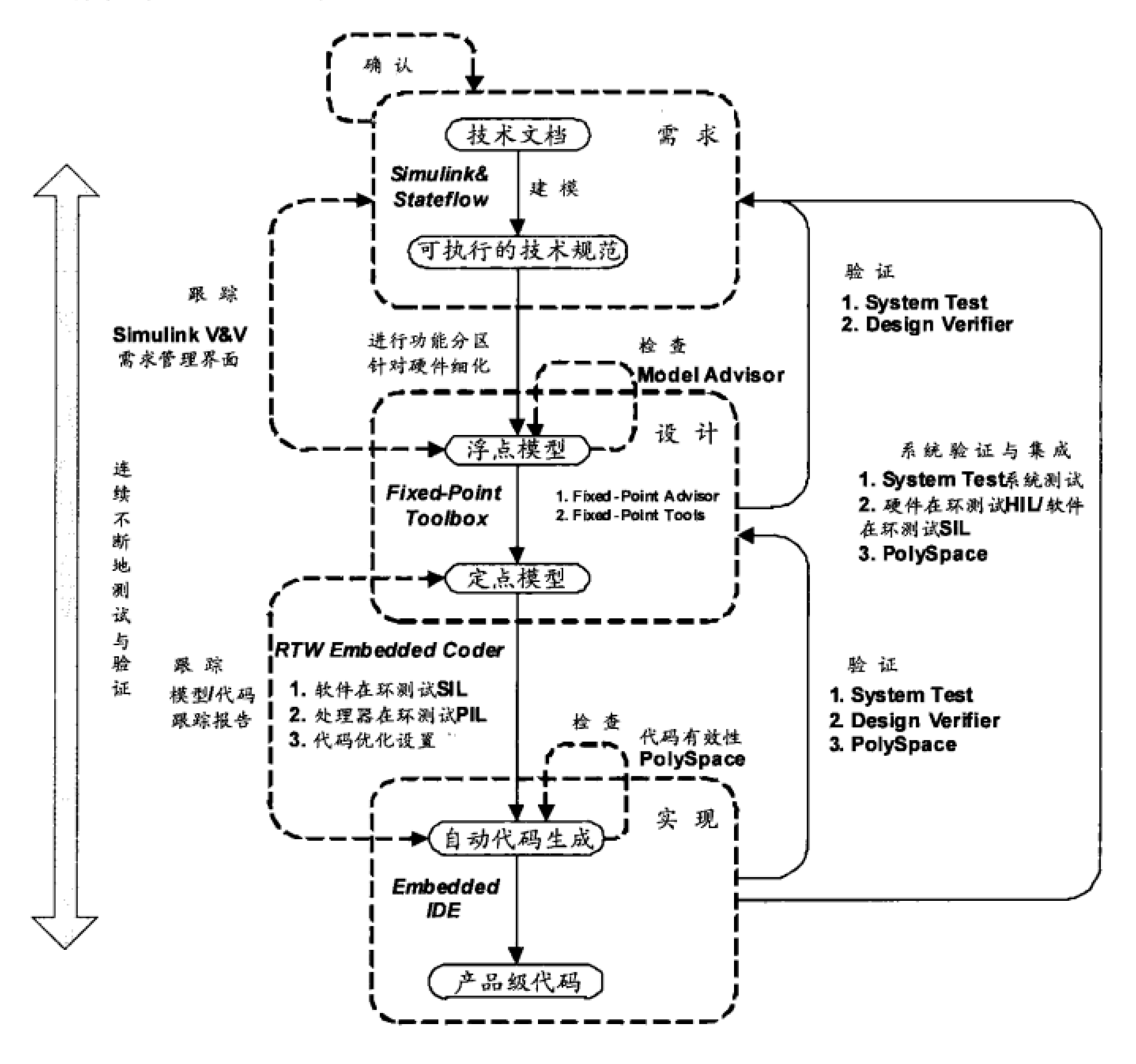


图 9.3.1 基于模型设计的工作流程

9.3.1 建立需求文档

系统设计人员在着手一个项目前，一般需要建立需求文档，这是项目管理和实现过程中不可或缺的重要部分。通常情况下，需求文档采用 Word、Excel、HTML、DOORs 等常用电子文档格式书写。

9.3.2 建立可执行的技术规范

在基于模型设计流程中，系统工程师会将需求文档转换成基于 Simulink&Stateflow 的可执行技术规范，以便对技术规范的内容进行验证与确认，实现对设计过程的早期验证，这也是

所谓的系统模型。可执行技术规范的建立为将来的算法确定、系统设计等提供了可靠的依据。

9.3.3 浮点模型

软件工程师根据需求文档中的算法要求、系统设计方案，对模型进行功能分区，并结合具体硬件的特点调整模型和算法，并对调整后的模型进行重新验证与确认，初步实现需求分析的要求。

9.3.4 需求与模型间的双向跟踪

基于模型的设计是一个连续不断地测试与验证的过程，在设计的每个过程中都有可能对模型进行优化和调整。这时，需求和模型间的双相跟踪就显得相当有必要了，双相跟踪可以确保需求与模型在设计的各个阶段保持正确关联。

利用需求与模型的双相跟踪，将文档中的每一条需求对应一个或多个模块关联，同时将模型中的每一个模块也对应一个或多个需求关联。在项目开发过程中，有时候会根据市场的变化增加或减少某些功能(需求)，若工程师发现某些需求根本不可达或需要进一步细化，可直接修改对应的模块，再通过需求一致性检查来追踪这些需求的变化，只要对这部分设计作适当的修改即可，而不会影响到整个的设计。同时可以发现缺失的需求与冗余等，及时根据需求修补缺陷，规避项目开发的市场风险。

9.3.5 Model Advisor 检查

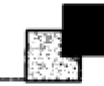
对前面建立的浮点模型，还需进行 Model Advisor 检查，这在设计早期特别有用。Model Advisor 能识别模型中隐含的问题，警告和限制代码效率的部分，找出模型设置是否会导致生成代码的无效或代码不符合安全标准。Model Advisor 可指导用户修补生成代码的模型，通过分析模型或模块的配置，给出模型组件的检查结果并提供模型的改进意见。

9.3.6 模型验证

模型验证是系统完整性需重点关注的问题，在基于模型设计的每一阶段都需对系统的设计进行检查、分析和测试工作，它贯穿于项目开发的全过程。一般主要进行以下几项测试：

(1) System Test 系统测试。System Test 为模型的测试提供了一个软件框架，工程师可以利用 System Test 预定义的元件创建测试程序，并运行和分析测试 MATLAB 代码或 Simulink 模型的结果。利用 MATLAB 脚本文件和分析工具对数据进行详细的分析与研究，保证了项目从研发到试生产阶段的测试过程都是标准与可重复的。

(2) Simulink Design Verifier 设计验证器。针对 Simulink 和 Stateflow 模型的验证，用户可采用 Simulink Design Verifier 自动生成测试用例，来满足模型覆盖度分析和用户自定义目标的要求，同时 Design Verifier 还可以检验模型的属性以及生成反例。用户也可在 Simulink 或 Stateflow 模型中自定义测试目标。使用属性检验功能，以寻找设计中的瑕疵、冗余的状



态、缺失的需求，这些问题在仿真过程中通常是很难发现的。

(3) 覆盖度分析。模型覆盖度是用来评估模型测试用例的累积结果，检验结果为一百分比，它表示以测试用例作为模型的输入信号，仿真结束后，仿真能到达的通路占所有通路的百分比。模型覆盖度检查记录下模型中每一个能直接或间接决定仿真通路的模块的执行情况，同时也记录模型中 Stateflow 图表的状态及状态转移情况。根据模型覆盖度报告，用户可以发现模型中是否存在从未被执行的功能模块，判断是模块冗余还是设计缺陷。

9.3.7 定点模型

上述仿真、检查、验证等一系列过程，已经在一定程度上证明了浮点模型的可行性，下面可针对具体的嵌入式处理器作定点化处理，使模型和算法在硬件上得到进一步优化，减少生成代码的长度以提高代码效率，降低功耗。

由于定点模型具有简化电路，缩小芯片体积，运算速度快，功耗低的优势，在基于模型设计的开发过程中，应尽量将模型中的数据类型作定点化处理。

用户可以自己手动设置定点数据类型，然后借助 Fixed-Point Tool 工具检查设置是否符合设计要求，或者用 Fixed-Point Advisor 工具自动定标，再借助 Fixed-Point Tool 工具优化定标。

9.3.8 软件在环测试(SIL)

软件在环测试(SIL)是在模型环境中，对模型自动生成的嵌入式 C 代码或手写代码进行非实时性联合仿真，以评估这些代码的优劣，完成对生成代码的早期验证。

软件在环测试不需要硬件，只是对算法代码进行测试，具体做法是对要进行测试的子系统编译可生成 SIL 模块，比较原模块与 SIL 模块的输出，以此确认算法的正确性。

9.3.9 处理器在环测试(PIL)

处理器在环测试(PIL)是将自动生成的 C 代码下载到处理器中，并和被控对象模型在模型中进行非实时仿真，即通过真实的 I/O、串口等来交换工作在处理器上的嵌入式 C 代码和运行在模型中被控对象模型间的数据，来评估代码在处理器上的运行过程，以便发现由目标编译器或处理器产生的错误，PIL 协同仿真可以帮助用户估计算法的优劣。通过 PIL 测试，可以在所支持的处理器上自动生成子系统的代码，还可以用原始的 Simulink 模型自动验证生成的嵌入式代码，包括在 Simulink 中无法仿真的一些目标特定的代码。

9.3.10 代码与模型间的双向跟踪

需求与模型间可建立双向跟踪，代码与模型同样也可以建立这样的跟踪，用户可以通过代码与模型间的链接，快速定位某个模块所对应的代码段，也可以通过分析代码，改进模型。

9.3.11 代码优化

为了消除 MATLAB 软件的一些瓶颈，对代码模型须按以下步骤进行修改：

(1) 进行 Model Advisor 检查并按其建议修正模型。

(2) 进行 DO－178b 航空电子规范检查(包括其他行业标准)(可选项)。

(3) 对模型按功能进行原子化处理。

(4) 对 RTWEC 进行优化设置(包括为具体芯片生成 C 代码，这可成倍提高代码效率)。

(5) 对生成的嵌入式 C 代码按传统方法进行优化。

9.3.12 生成产品级代码

经过上述一系列的验证和优化，生成的 C 代码可和手写代码媲美。

9.4 需求分析及跟踪

9.4.1 系统模型

根据 PID 控制电动机这项任务的要求，模型中需要以下基本功能：

- 电动机控制信号输入。
- PID 控制器。
- 直流电动机模型。
- 转速监控。

按照这些需求，建立需求文档，如图 9.4.1 所示。

电动机 PID 控制系统需求

1. Control Signal Input

控制信号输入

2. PID Controller

PID 控制器

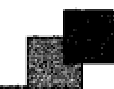
3. DC Motor

直流电动机模型

4. 4. Monitor

转速监控

图 9.4.1 需求文档



根据上述需求建立模型，这里将使用在第 2 章建立的直流电动机模型，如图 9.4.2 所示。

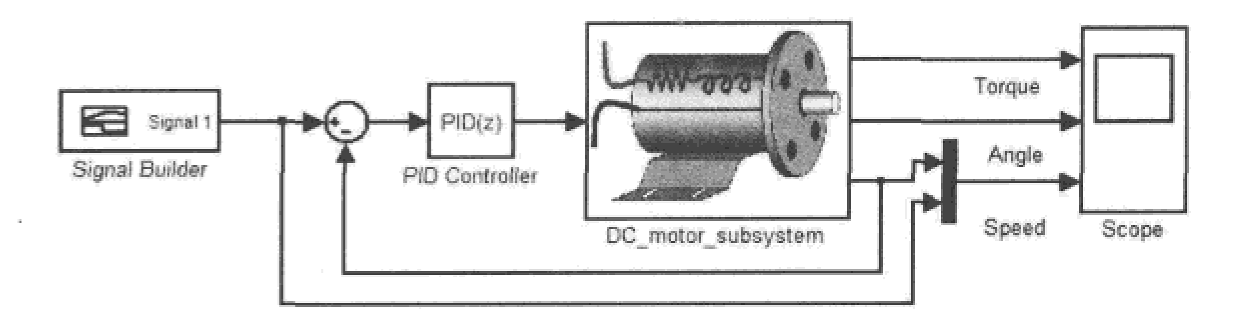


图 9.4.2　直流电机模型

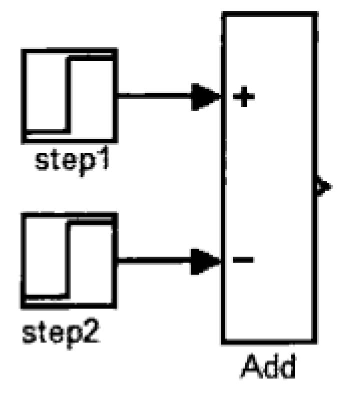


图 9.4.3　信号发生模块组

为了使其满足文档中的需求，显然还需要做以下修改：

(1) 用图 9.4.3 所示的两个 step 模块和 add 模块代替 Signal Builder 模块作为速度控制信号（Signal Builder 模块不支持后续过程中的某个步骤），将输出信号幅度调整为 225，并适当延长控制信号的持续时间，以求尽量接近实际。

其中，step1 模块的阶跃时间（step time）设为 10 s，信号幅度（Final Value）设置为 225；step2 模块的阶跃时间设置为 50 s，信号幅度设置为 225。

然后将这组模块创建为一个子系统，作为速度控制信号输入，如图 9.4.4 所示。

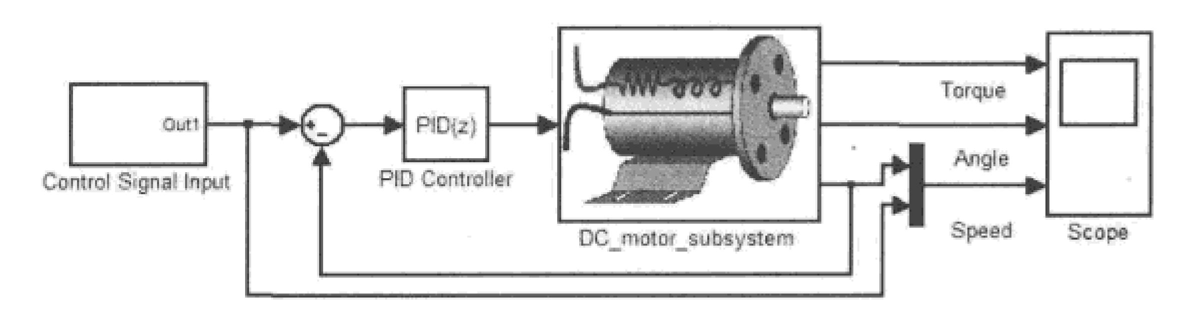


图 9.4.4　更换控制信号模块

修改后的输出信号波形如图 9.4.5 所示。

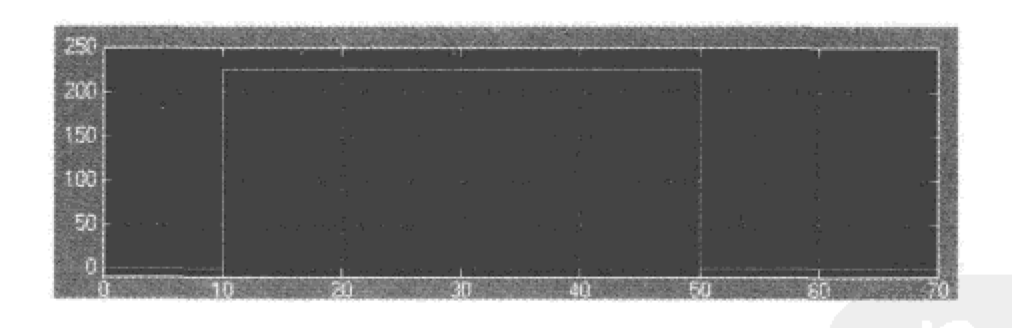


图 9.4.5　输出信号模型

(2) 将 Sum 和 PID 模块创建为一个子系统，对应需求中的 PID 控制器，如图 9.4.6 所示。

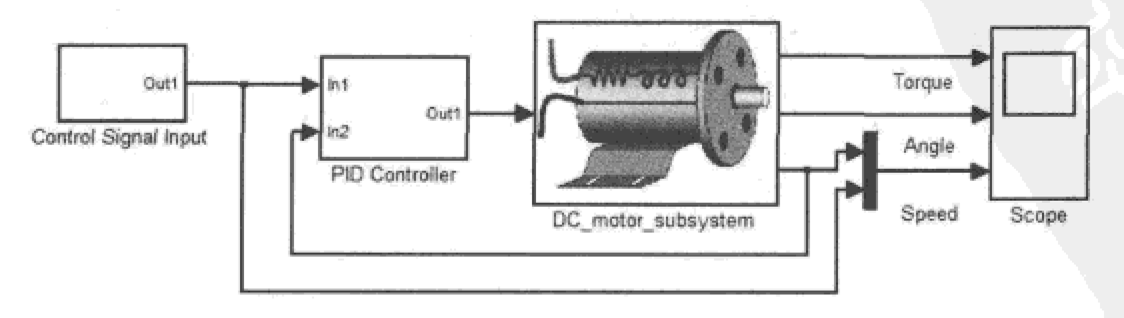


图 9.4.6　创建 PID 子系统



(3) 修改 DC_motor_subsystem 子系统，删去转矩和角度输出，仅关心其转速值，对应需求中的直流电动机模型，将电动机参数调整为 $R=2, L=0.1, K_m=0.1, K_f=0.1, K_{emf}=0.1, J=0.1$。子系统修改后如图 9.4.7 所示。

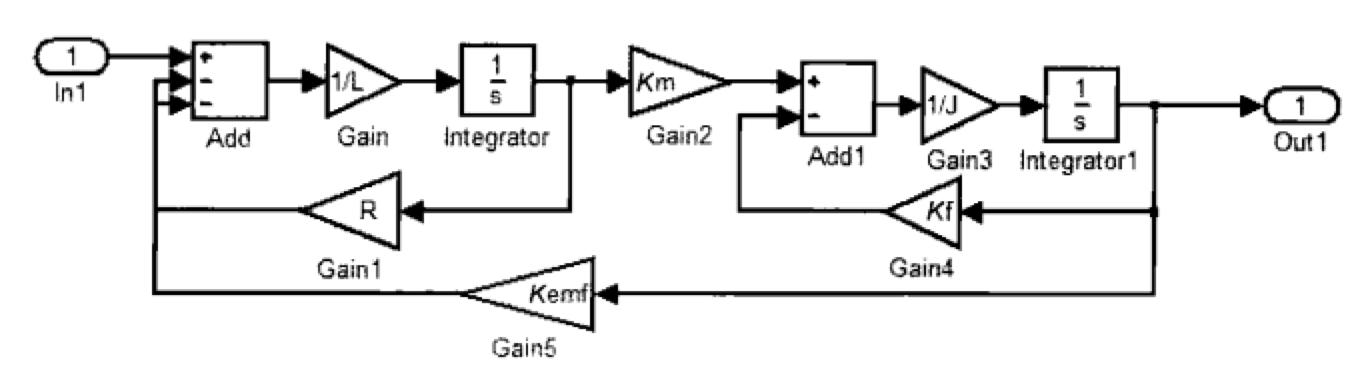


图 9.4.7　DC_motor_subsystem 子系统

(4) 完成第三步后，将示波器由 3 通道改为 2 通道。电动机模型仅余一个输出口(电动机转速)，将其与 Control Signal Input 信号通过合路器 Mux 模块一并连接到示波器上，以监视其响应，对应于需求中的转速监控。同时，可以将速度控制信号和 PID 输出信号也连接到示波器上，方便分析三者间的关系，如图 9.4.8 所示。

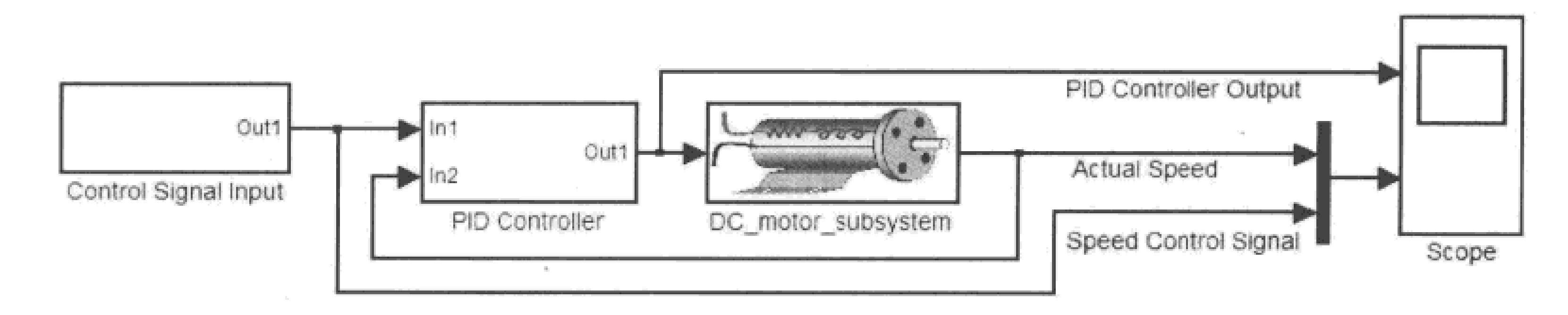


图 9.4.8　功能验证模型

考虑到最后的嵌入式实现，将 PID 模块的算法修改为 PI，然后按照第 2 章介绍的方法调整其 P、I 参数，得到合适的值。可以看到电动机实际转速的响应曲线(A)和控制信号拟合度较好，PID Controller 模块的输出范围为－700～＋1150，如图 9.4.9 所示。

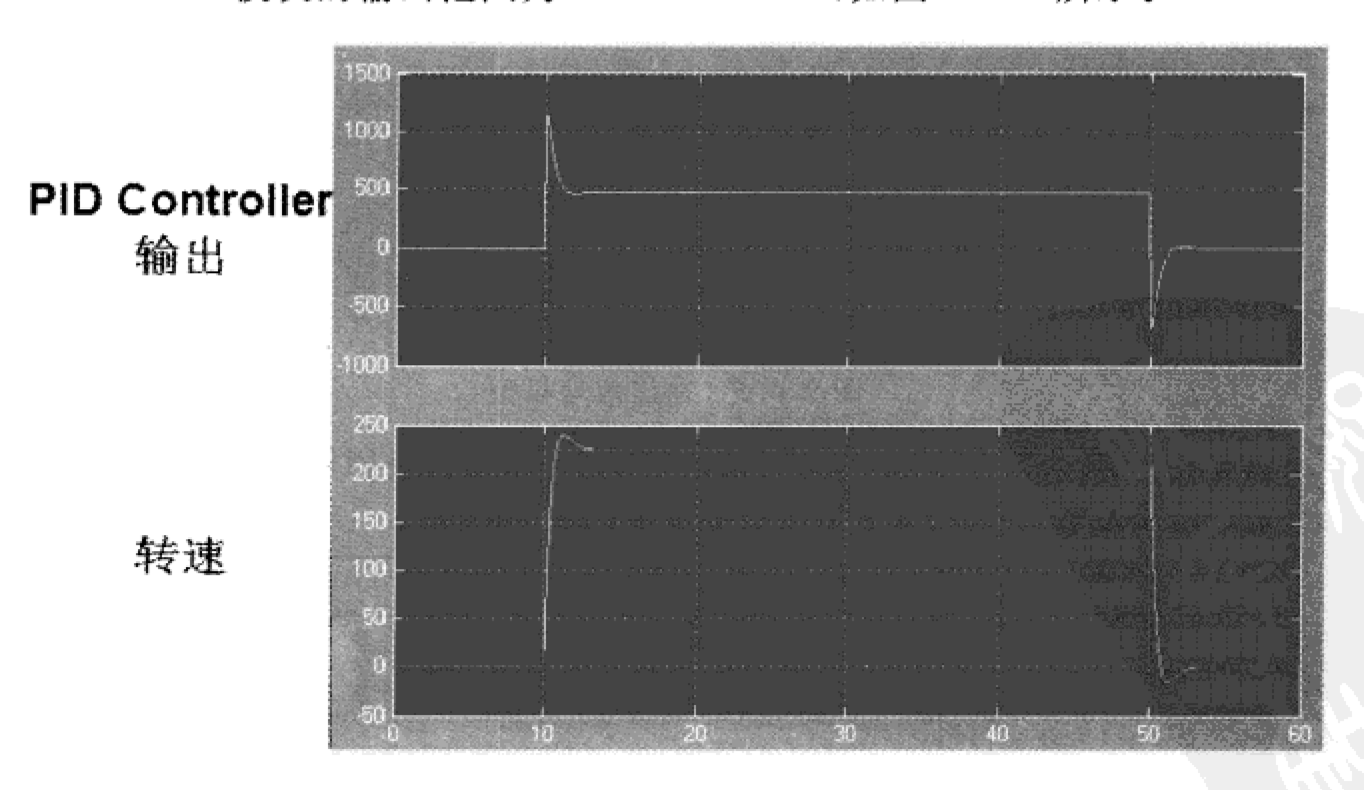


图 9.4.9　仿真波形



9.4.2　需求关联

1. 注册控件

在建立需求到模块的关联之前，需要首先注册 ActiveX 控件，以便在需求文档（以 word 为例）里加入模块导航按钮。过程如下：

```
>> rmi setup
Registered the requirements Active-X controls
>>
```

2. 关联选项

选择模型窗口的菜单项 Tools → Requirements → Setting...，打开需求设置选项。

Report 页，用户可以设置需求报告所包含的内容。如表 9.4.1 所列。

表 9.4.1　报告选项

Highlight the model before generating report	在生成报告前，高亮显示已关联需求的 Simulink 模块，并在体现在生成的报告里
Report objects with no links to requirements	列出未关联到需求的 Simulink 模块
Show User tags for each reported links	在每项报告链接中显示用户标签
Use document index in requirements tables where possible	如果需求文档的 ID 可用，则显示在报告需求表格的 ID 栏显示为代替文档路径
Include links to object	含有至目标的链接

在 Selection-Based Linking 界面中，用户可以设置需求关联形式、文档位置等，这里仅简要说明各个选项的意义，使用技巧用户需要自行体会。初期建议将模型与需求文档放在同一个目录，并按图 9.4.10 进行设置 Selection-Based Linking 界面。熟练掌握后，则可按项目管理的要求，修改 Selection-Based Linking 界面设置，并安排需求文档及模型的正确位置。

图 9.4.10　Selection-Based Linking 界面设置

表 9.4.2　设置项含义

Enabled Applications	在将模块关联到需求文字时，选择可用的链接类型 如图 9.4.10 选中 Word 与 Excel，右击模块的 Requirement 菜单，会出现两个选项 Add link to Word selection、Add link to active Excel cell
Document file reference	指定需求文档的位置 • absolute path：绝对路径 • path relative to current directory：当前目录 • path relative to model directory：模型存放的目录 • filename only (on MATLAB path)：仅文件名
Modify documents to include links to models	使能该选项，当完成需求关联后，在对应的需求文字后加入导航按钮，默认以书签形式插入
Model reference	指定从需求文档寻找关联模型的路径 • absolute：绝对路径 • none (on MATLAB path)：相对路径

3. 建立关联

本文以 Microsoft Word 作为文档编辑环境，建立图 9.4.1 所示的需求文档，由于目前 MATLAB 诸多组件不支持中文，因此需要关联到模块的需求主题词，必须以英文表述。

建立关联的步骤如下：

(1) 在需求文档中，选中需要建立关联的主题词；例如选中 1. Control Signal Input。

(2) 右击需要建立关联的模块，如 Control Signal Input 子系统模块，选择菜单项 Tools → Requirements → Add link to Word selection，如图 9.4.11 所示。

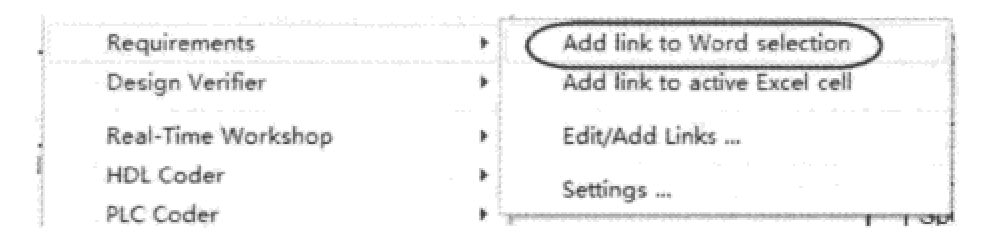


图 9.4.11　建立关联

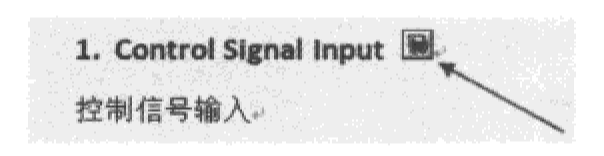


图 9.4.12　文档中显示关联图标

(3) 这时需求主题词的末尾，加入了一个导航按钮，如图 9.4.12 所示。其他需求可按同样方法与模块相关联。

(4) 再次右击先前的模块，选择菜单项 Tools → Requirements → 1. Control Signal Input，则对应的需求主题词高亮显示，如图 9.4.13、图 9.4.14 所示。

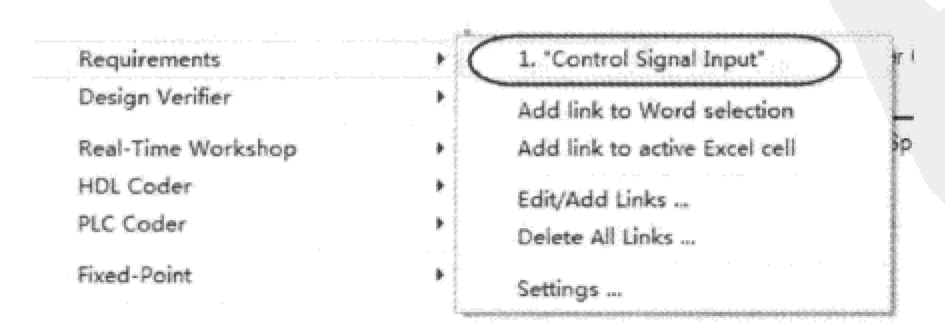
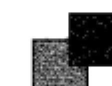


图 9.4.13　模块关联的需求文档



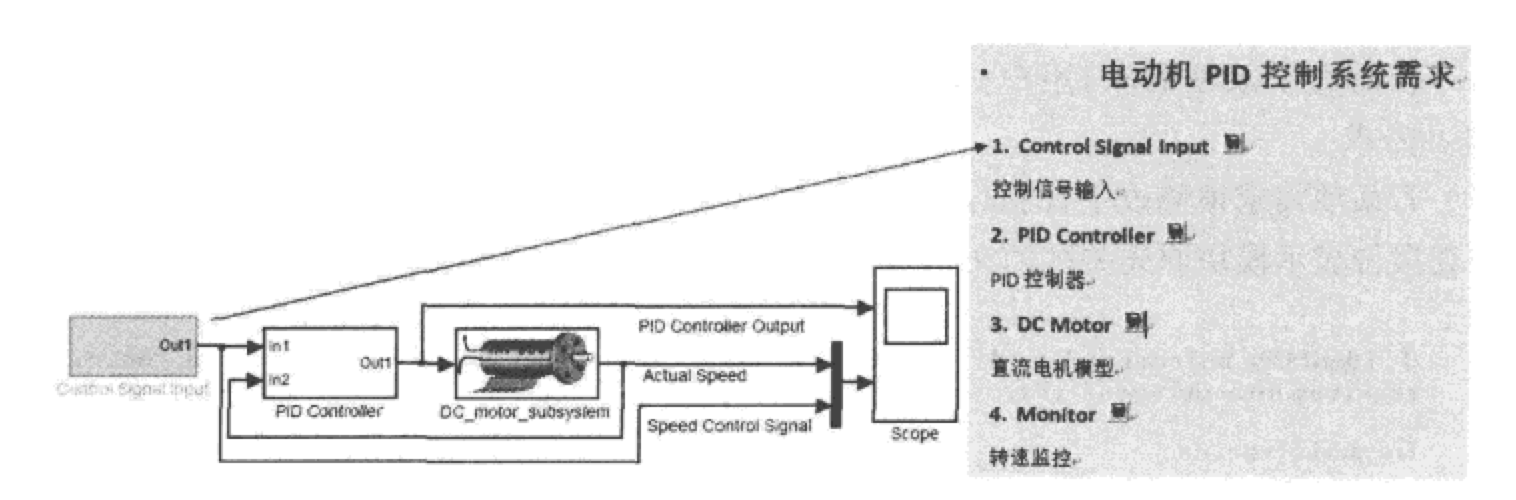


图 9.4.14　关联模块高亮

(5) 双击需求主题词末尾的导航按钮，对应的模块高亮显示，说明需求与模块之间已建立了关联，如图 9.4.15 所示。

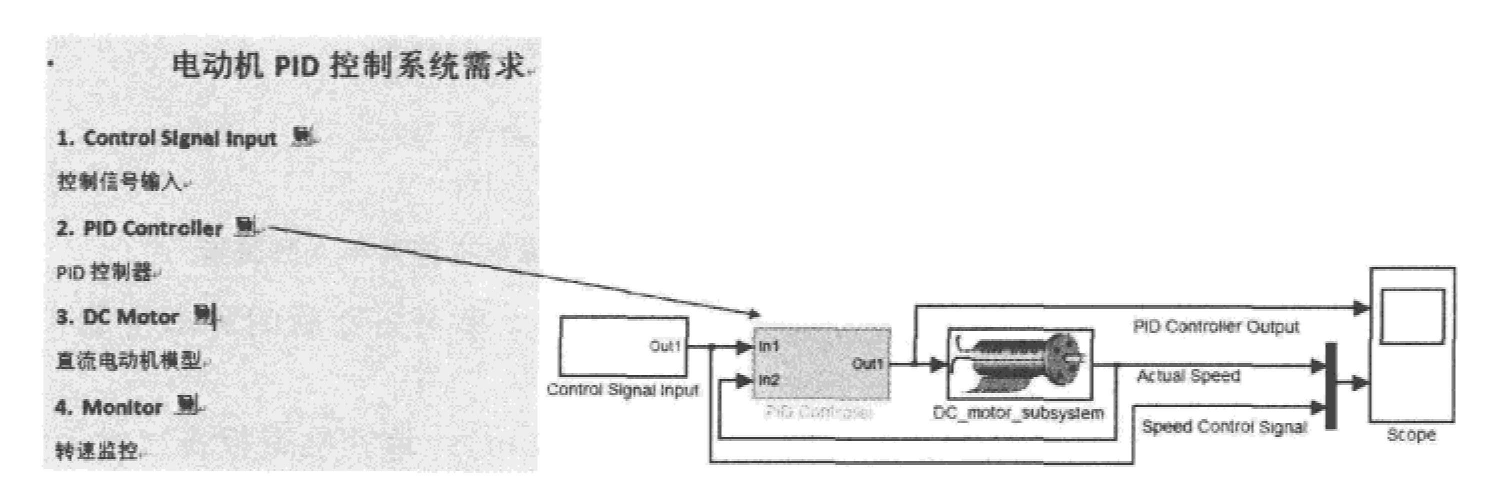


图 9.4.15　文档关联到模型

9.4.3　一致性检查

1. 完成文档与模型间的关联后，可立即进行一致性检查

(1) 选择模型窗口菜单项 Tools → Requirements → Consistency checking...，系统打开 Model Advisor 界面，如图 9.4.16 所示，并自动选中 Requirements consistency checking 子类。

Report name: Model Advisor - Requirement consistency checking

Simulink version: 7.6　　Model version: 1.19

System: DC_MOTOR_mbd　　Current run: 07-Oct-2010 22:01:18

Run Summary



Pass	Fail	Warning	Not Run	Total
4	0	0	0	4

图 9.4.16　Model Advisor 界面

(2) 关闭 Word 应用程序，再单击右侧的 Run Selected Check 按钮，开始检查。

(3) 检查完成，报告显示通过一致性检查。

2. 需求变更

(1) 若增加了某项需求,用户只要查看文档是否存在尚未关联的需求主题词,即可判断是否增加了需求。

(2) 若某项需求被修改了,例如,将文档中的 1. Control Signal Input 修改为 1. Control Signal,一致性检查报告显示模块原先关联的文字与当前文档中的需求主体词不一致,如图 9.4.17 所示。

⚠ **Identify selection-based links having description fields that do not match their requirements document text**

Inconsistencies:

The following selection-based links have descriptions that differ from their corresponding selections in the requirements documents. If this reflects a change in the requirements document, click **Update** to replace the current description in the selection-based link with the text from the requirements document (the external description).

Block	Current description	External description	
DC_MOTOR_mbd/Control Signal Input	Control Signal Input	Control Signal	Update

图 9.4.17 模块原先关联的文字与当前文档中的需求主体词不一致警告

单击报告右下角的 Update 链接,可将模块原先关联的文字用更新后的需求主题词代替。再次执行该项检查,即显示通过。

(3) 若某项需求被删除,例如删去 1. Control Signal Input,一致性检查报告显示无法定位到先前关联的需求主体词,如图 9.4.18 所示。

⚠ **Identify requirement links that specify invalid locations within documents**

Inconsistencies:

The following requirements link to invalid locations within their documents. The specified location (e.g., bookmark, line number, anchor) within the requirements document could not be found. To resolve this issue, edit each requirement and specify a valid location within its requirements document.

Block	Requirements
DC_MOTOR_mbd/Control Signal Input	Control Signal Input

图 9.4.18 无法定位到先前关联的需求主体词警告

单击报告左下角的链接 DC_MOTOR_mbd/Control Signal Input,可定位到对应模块,单击右下角的链接 Control Signal Input,可打开该模块的需求编辑窗口。用户可根据需要删除模块或重新关联需求。

3. 模块变更

(1) 在后续的仿真过程中,若需要新增模块,用户需自行将新模块关联到对应的需求主题词。例如,在模型中添加一个 Constant 模块,在模型窗口中选择菜单项 Tools → Requirements → Generate report,在设置了 Report objects with no links to requirements 选项的前提下,如果模型存在未关联到需求的模块,报告会列出相应的模块,如图 9.4.19 所示。

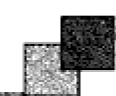


Table 3.2. Objects in "DC_MOTOR_mbd" that are not linked to requirements

Name	Type
Constant	Constant

图 9.4.19　未关联的模块

另外，选择菜单项 Tool → Requirements → Highlight model，关联了需求的模块会被高亮显示，由此可作快速分辨，如图 9.4.20 所示。

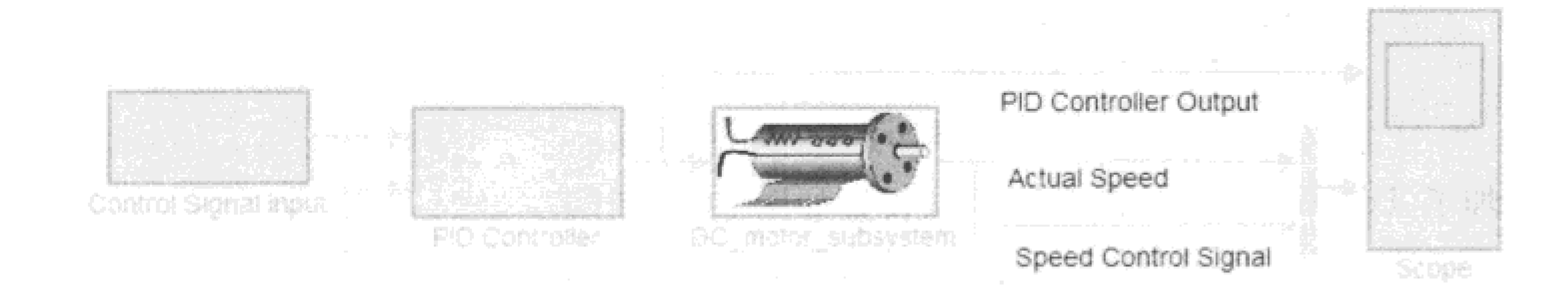


图 9.4.20　关联模块高亮

（2）若用户径自删除了模型里的某个模块，用户可逐一单击需求导航按钮来确定某个主题词已无对应的模块。当然恰当的办法是在删除模块前，先链接到需求文档，修改文档后再修改模型。

9.5　模型检查及验证

9.5.1　System Test

SystemTest 为用户提供了一个框架，在同一个环境下可以集成软件、硬件、仿真以及其他类型的测试。用户使用预定义的元素，方便快捷地创建并维护测试程序，之后保存并共享它们，这样在整个开发过程可重用这些测试程序，保证了测试是标准的。

SystemTest 软件集成了数据管理与分析功能，能够保存测试结果，实现了基于模型设计所要求的——连续不断的测试。

SystemTest 特点，如表 9.5.1 所列。

表 9.5.1　SystemTest 特点

类　型	特　点
图形化环境	用户可以使用图形化测试开发环境，快速建立测试程序
可重现的测试	所有使用 SystemTest 开发的测试程序，均可共享使用
参数测试	建立参数测试向量，可对模型进行迭代测试
可维护性	因为开发测试程序使用的是图形化的界面，因此用户不需要了解复杂的代码，快速修改这些程序

(1) 模型调整。根据本例的特点，选择电动机模型中的参数 K_f(即摩擦因数)与电动机模型的响应曲线作为测试对象，将电动机实际转速信号保存到工作空间，模型调按图 9.5.1 调整。

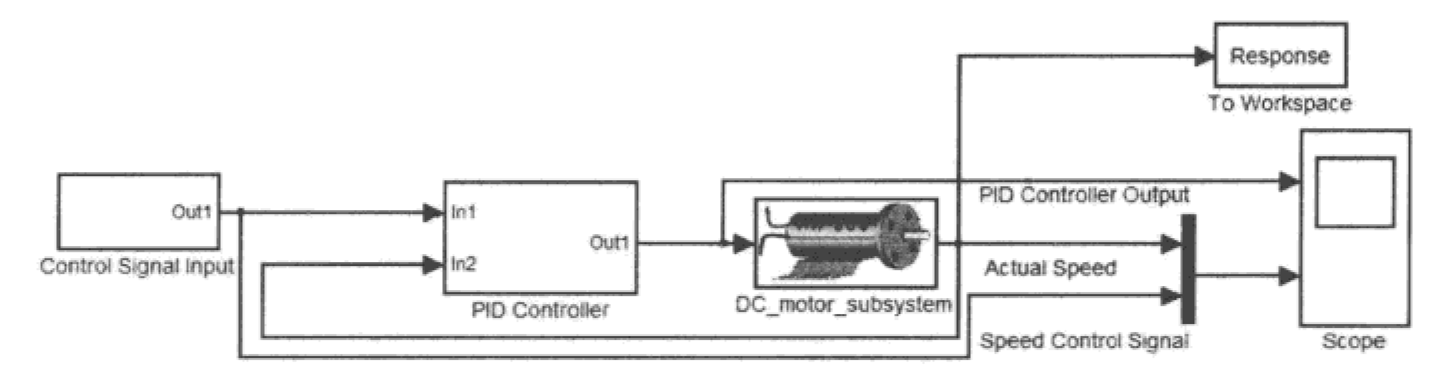


图 9.5.1 调整模型

增加了 To Workspace 模块，该模块设置如图 9.5.2 所示。

```
Parameters
Variable name:
Response
Limit data points to last:
inf
Decimation:
1
Sample time (-1 for inherited):
-1
Save format: Array
☐ Log fixed-point data as an fi object
```

图 9.5.2 To Workspace 模块设置

(2) 建立测试程序。在 MATLAB 命令行窗口输入 systemtest，打开 SystemTest 界面如图 9.5.3 所示。

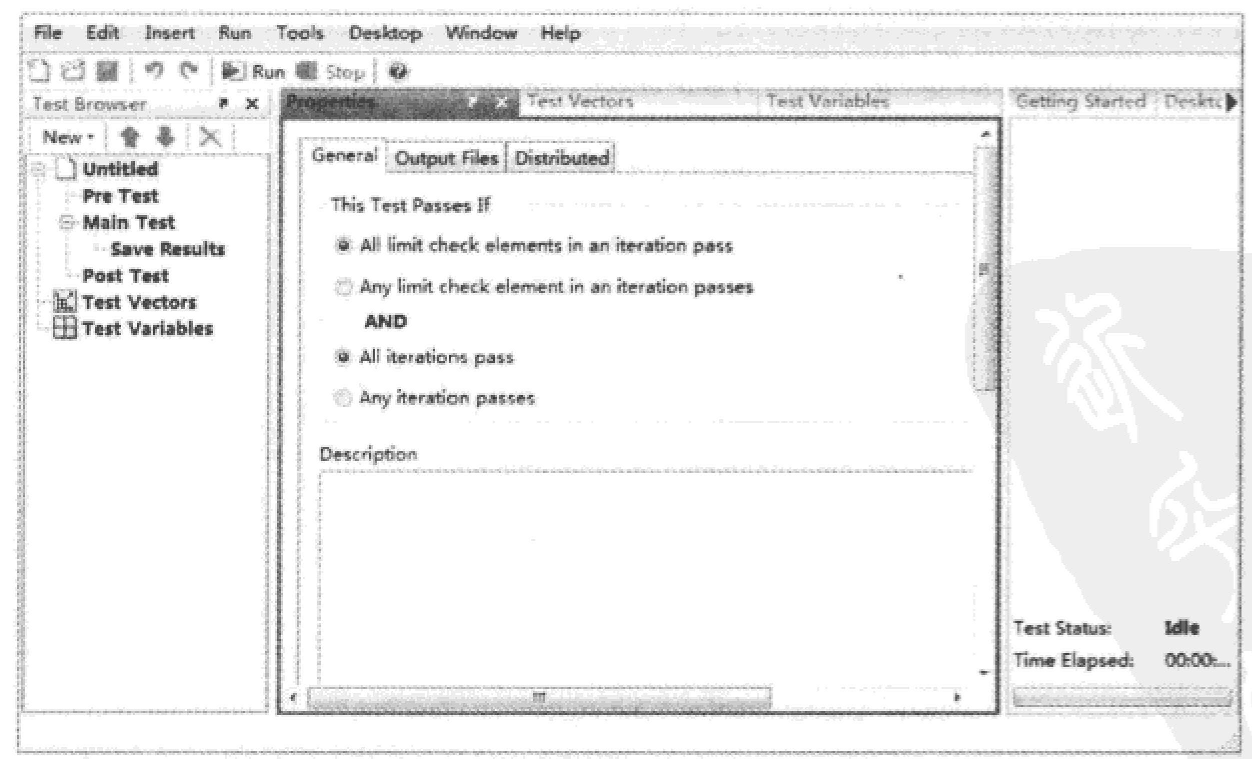
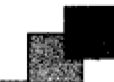


图 9.5.3 SystemTest 界面



(3) 添加测试模型。选择 SystemTest 菜单项，Insert → Test Element → Simulink，如图 9.5.4 所示。

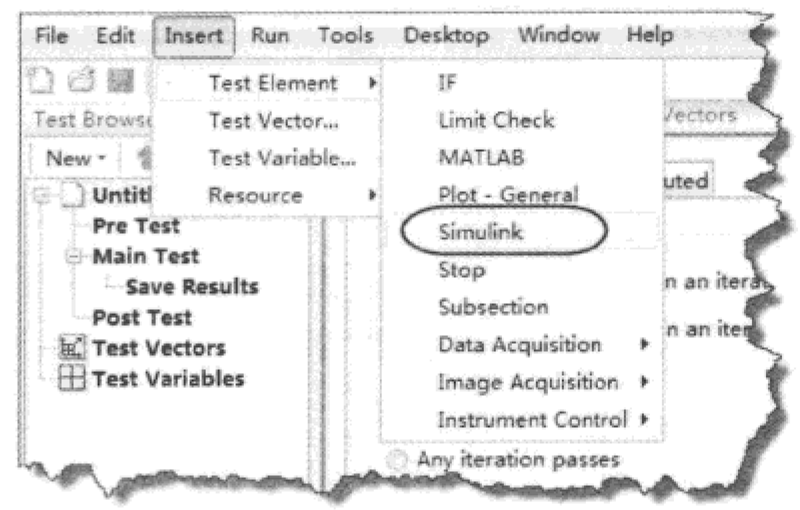


图 9.5.4　添加测试模型

在 Simulink model 文本框里输入待测试的模型路径，或单击右侧的 Browse... 按钮，指定模型路径，如图 9.5.5 所示。

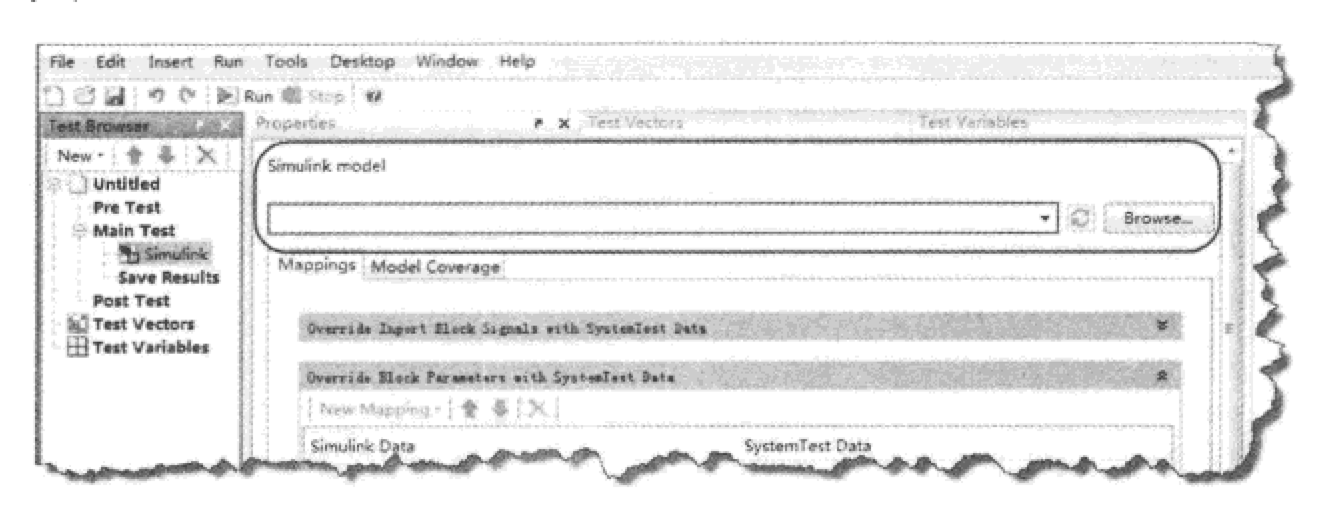


图 9.5.5　指定模型路径

(4) 添加测试参数向量。在中间界面的 Mappings，展开第二设置项 Override Block Parameters with System Test Data，增加一个新的参数映射，选择 Select Block to Add... 选项，如图 9.5.6 所示。

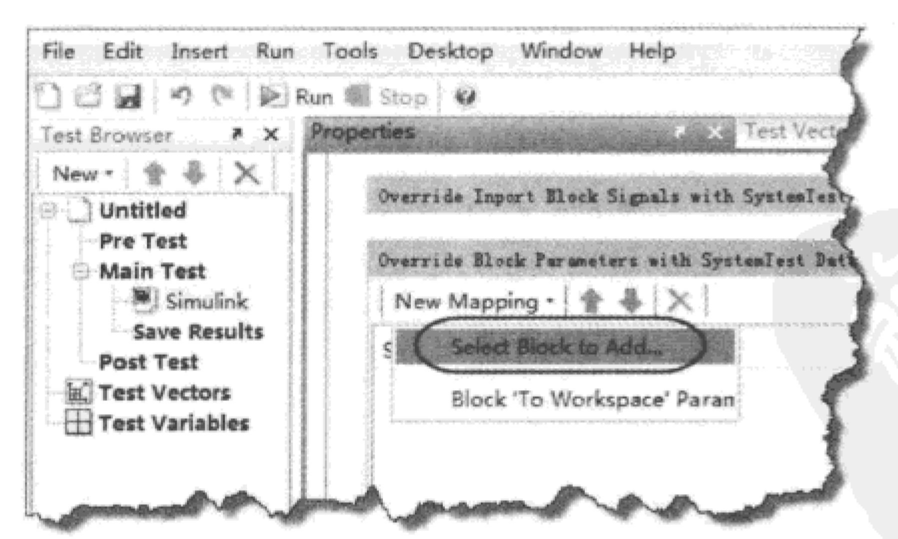


图 9.5.6　增加参数映射

在随后打开的模型窗口单击 To Workspace 模块，之后返回 System Test 主窗口。

在新增的参数映射栏 Simulink Data 列表，选择 DC_motor_subsystem ：Viscous Friction

Constant 选项，在 SystemTest Data 列表中，选择 New Test Vector... 选项，如图 9.5.7 所示。

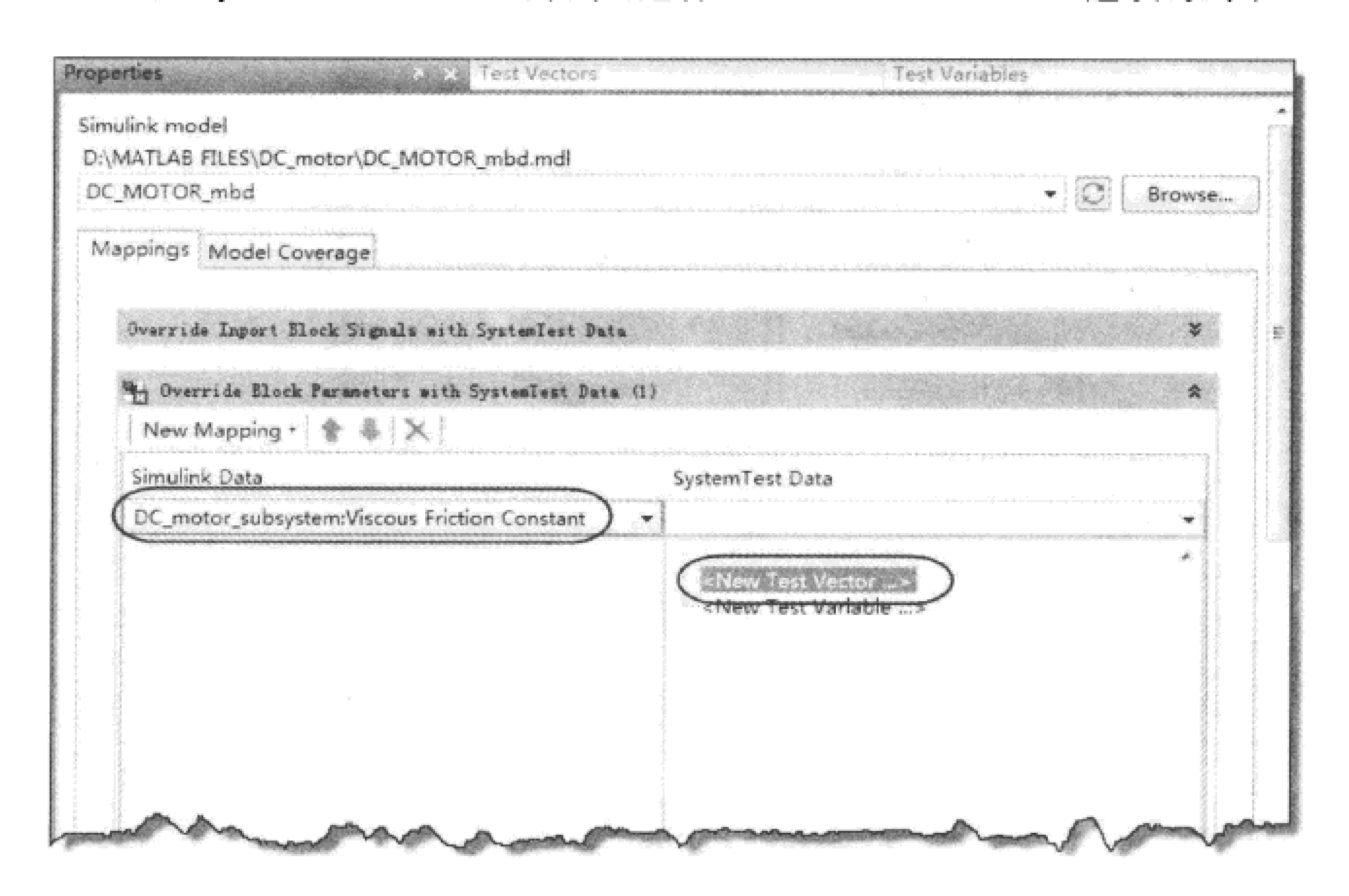


图 9.5.7 新增测试向量

在随后打开的 Insert Test Vector 对话框里，为向量命名，如 Kf；并为其指定取值范围，如[0 : 0.1 : 1]，如图 9.5.8 所示。

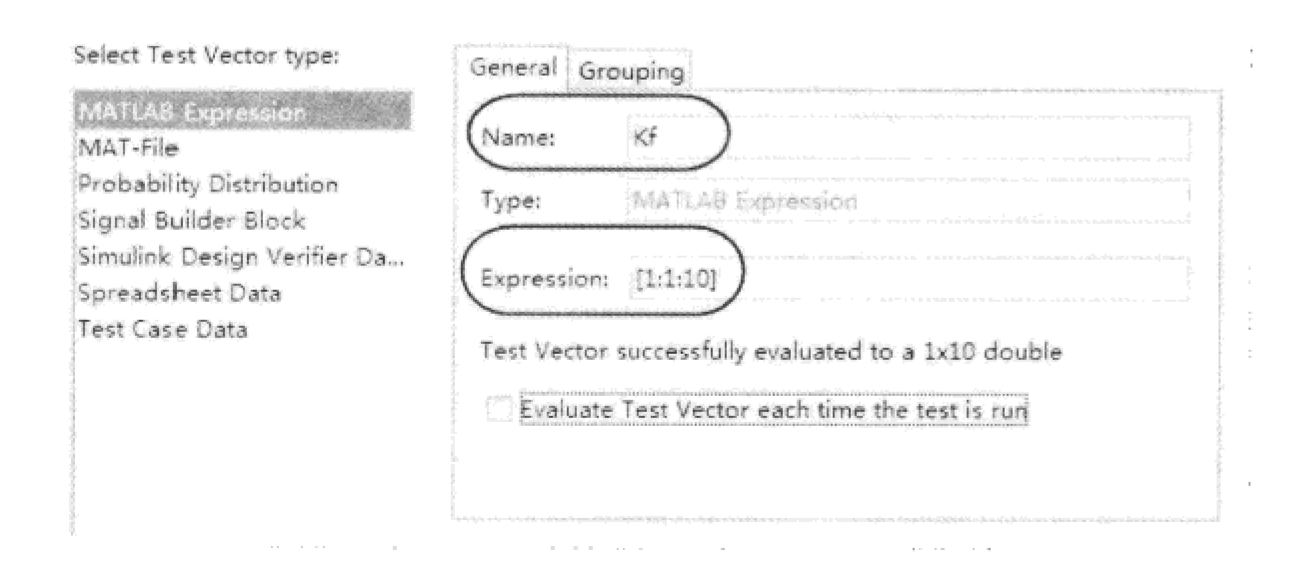
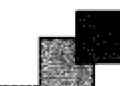


图 9.5.8 测试向量命名

(5) 添加输出变量。在中间页面的 Mappings 列表中，展开第 5 设置项 Assign Model Outputs to SystemTest Data，增加一个新的输出信号映射，选择 To Workspace Block 选项，如图 9.5.9 所示。

在新增的输出映射栏 Simulink Data 列表中，选择 To Workspace 选项，此即对应着模型里的 To Workspace 模块；在 SystemTest Data 列表中，选择 New Test Variable... 选项，如图9.5.10 所示。

在随后打开的 Edit To Workspace 对话框里，为变量命名，例如 To Workspace，如图 9.5.11 所示。



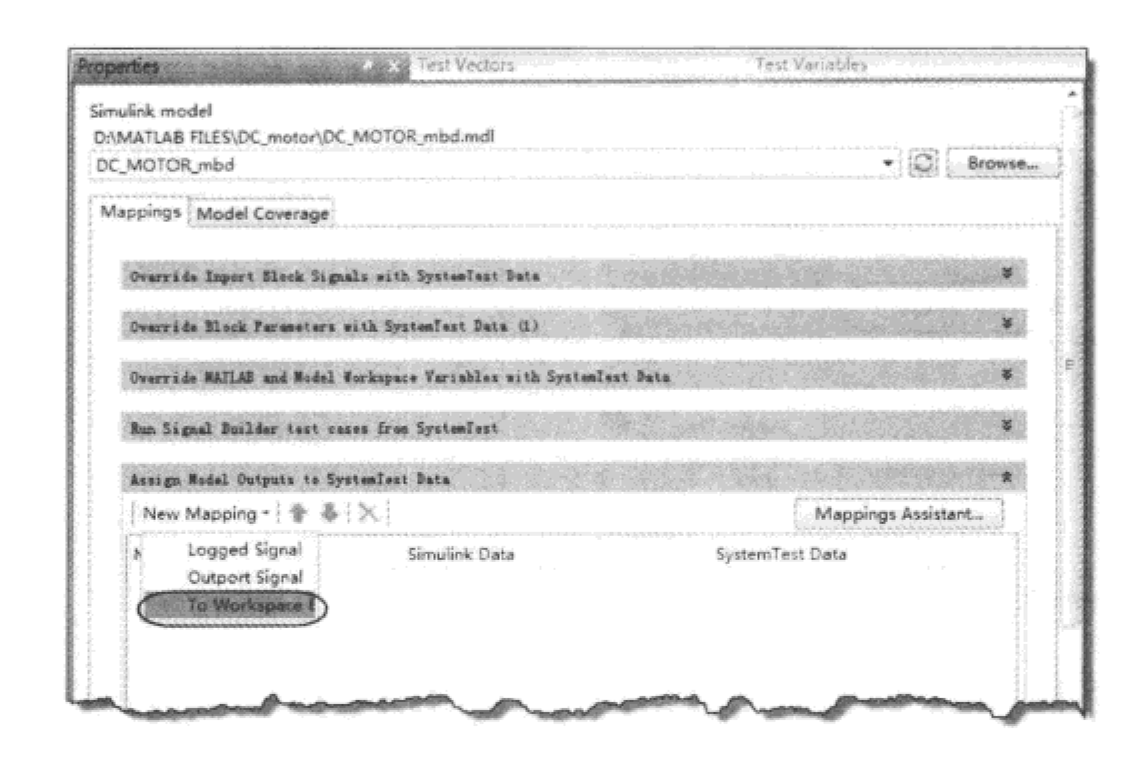


图 9.5.9 增加输出信号映射

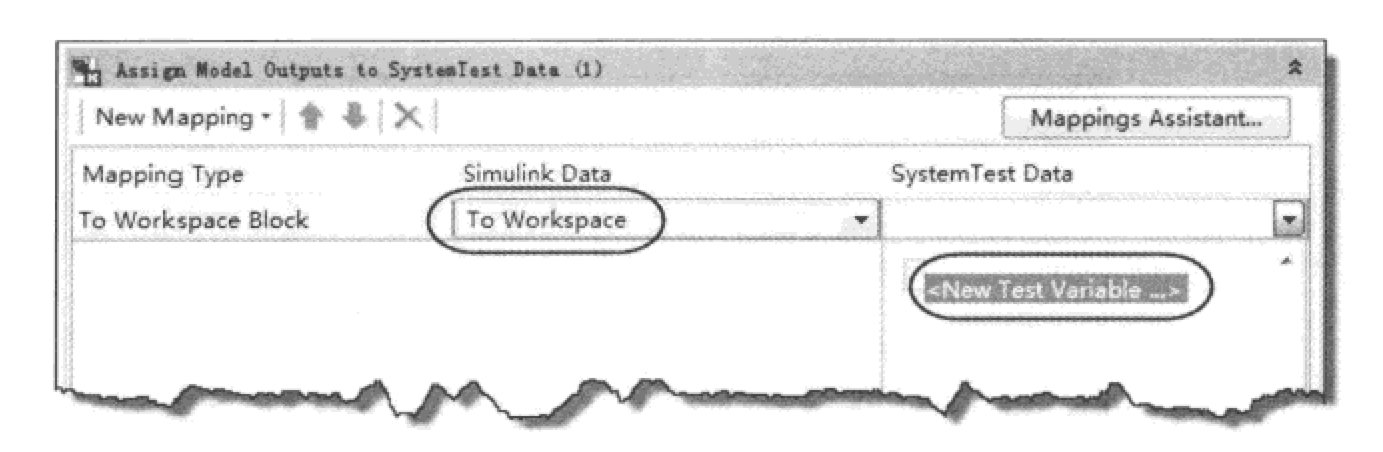


图 9.5.10 新增测试变量

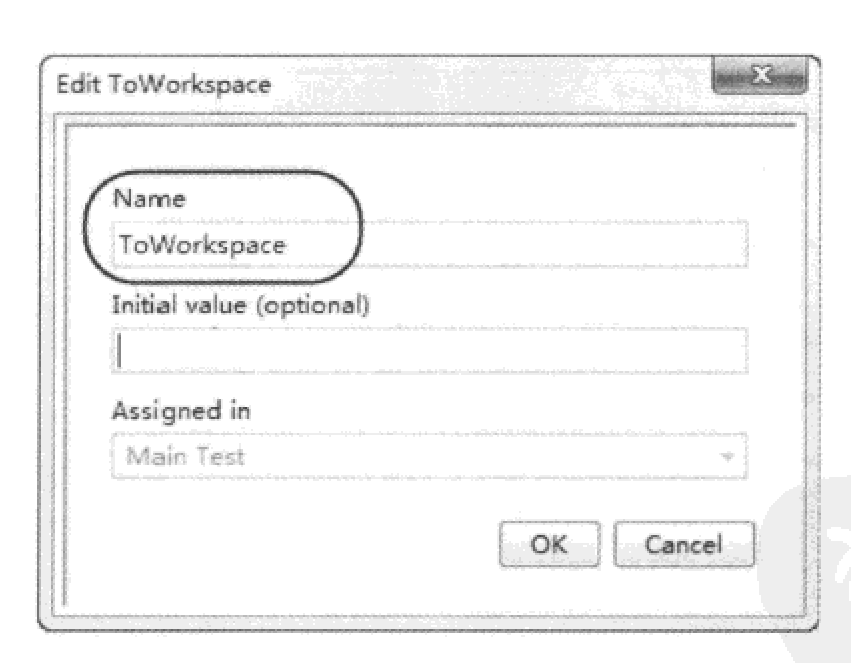


图 9.5.11 变量命名

(6) 添加绘图测试。本测试的目的是为了得到不同参数条件下的系统响应,添加绘图测试元素后可直观地显示响应曲线。

选择 SystemTest 菜单项,Insert → Test Element → Plot-General,如图 9.5.12 所示。

单击 add plot 按钮在下拉菜单中选择 plot 选项,添加绘图类型,如图 9.5.13 所示。

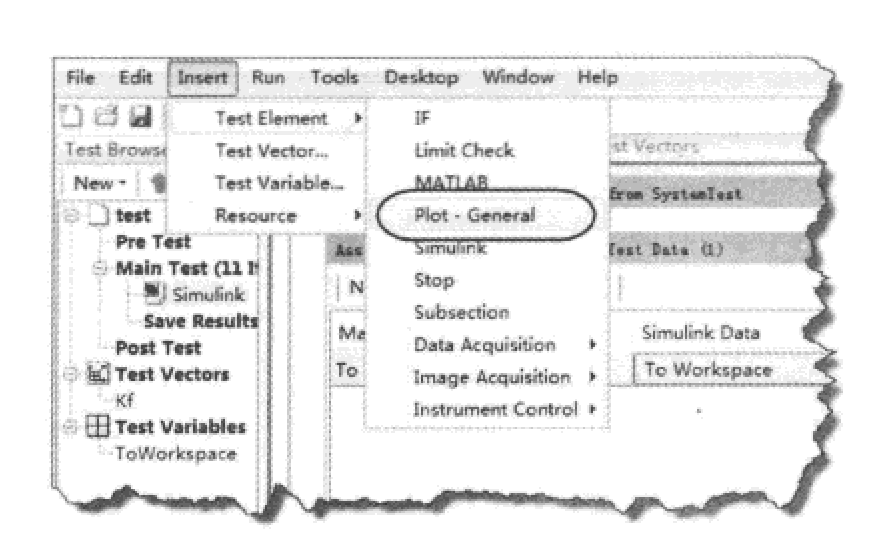


图 9.5.12 添加绘图测试

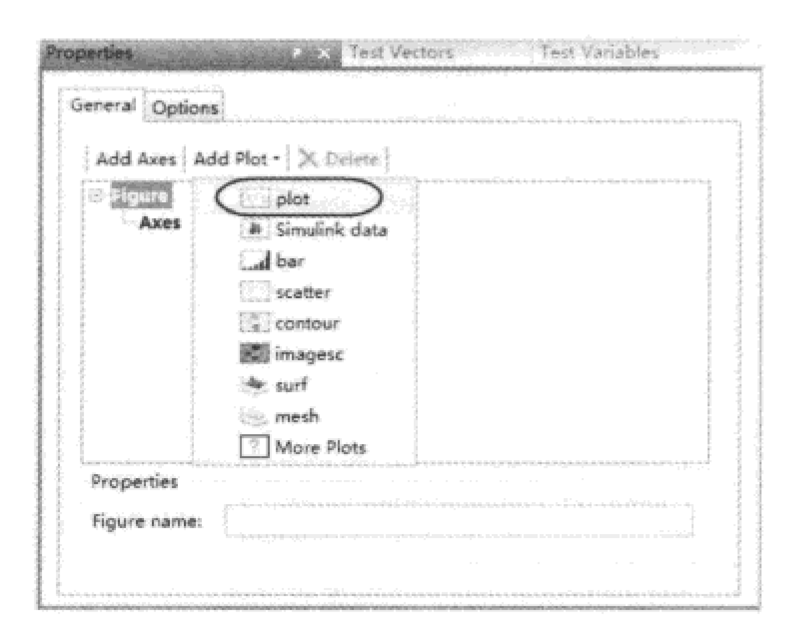


图 9.5.13 添加 plot 绘图类型

添加 plot 绘图类型后，在 Y Data Source 列表框中选择 To Workspace 选项，作为图形的纵坐标参数，如图 9.5.14 所示。

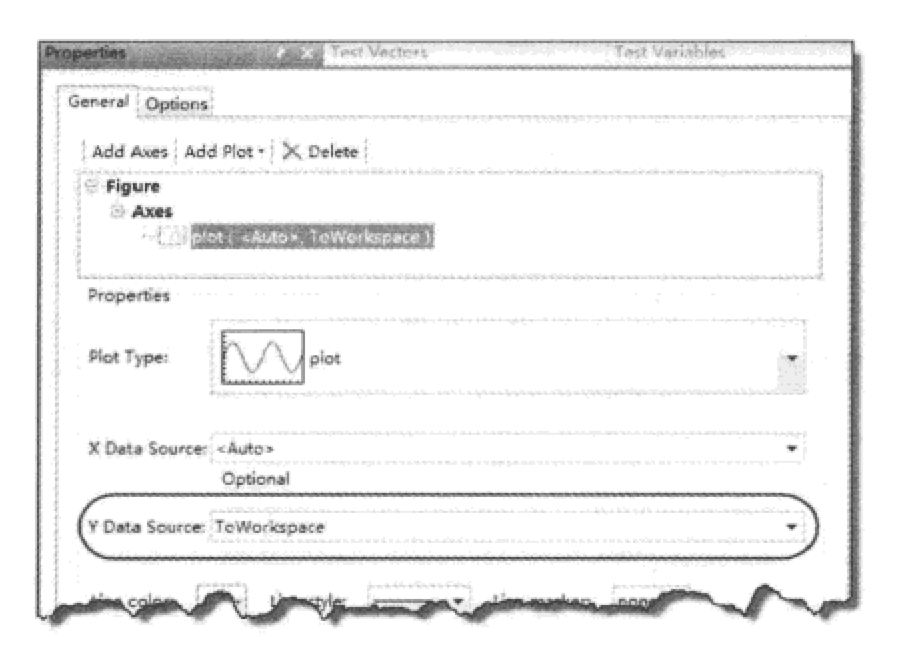


图 9.5.14 指定纵坐标数据

在 Options 选项卡中，选择 keep any existing data on the figure 选项，如图 9.5.15 所示。

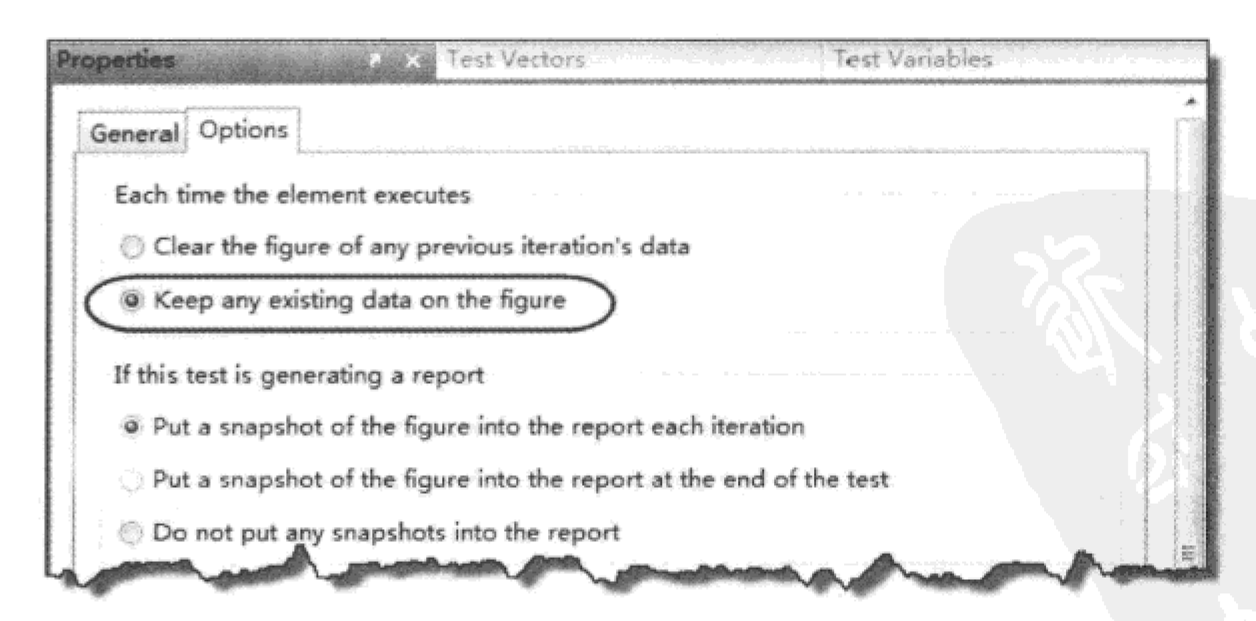
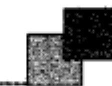


图 9.5.15 Options 选项卡



（7）添加输出结果。选中 System Test 主窗口的左侧窗口的 Save Results 选项，在中间窗口单击 New Mapping 按钮，新增输出结果映射，如图 9.5.16 所示。

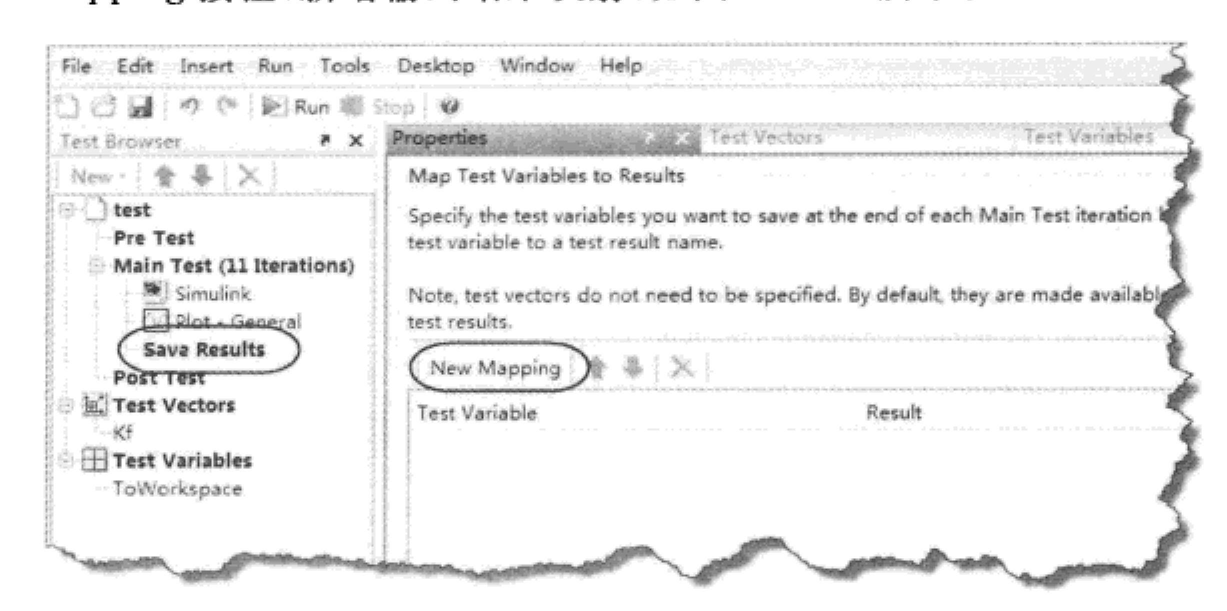


图 9.5.16　添加输出结果

根据需要，选择输出结果 To Workspace，并取名为 To Workspace，如图 9.5.17 所示。

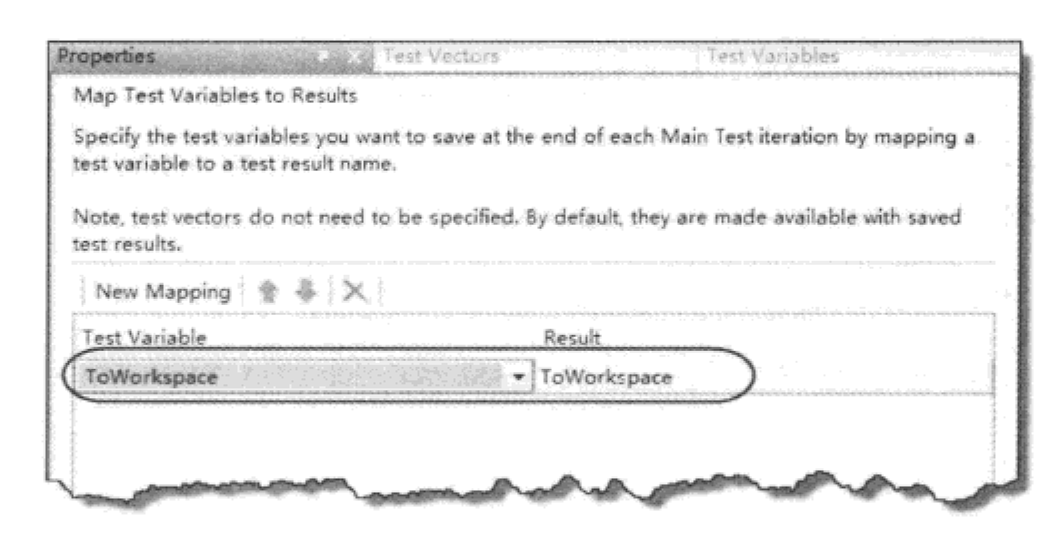


图 9.5.17　选择输出结果

（8）执行测试并观察结果。保存测试程序，并单击菜单栏的 Run 按钮，开始测试系统，如图 9.5.18 所示。

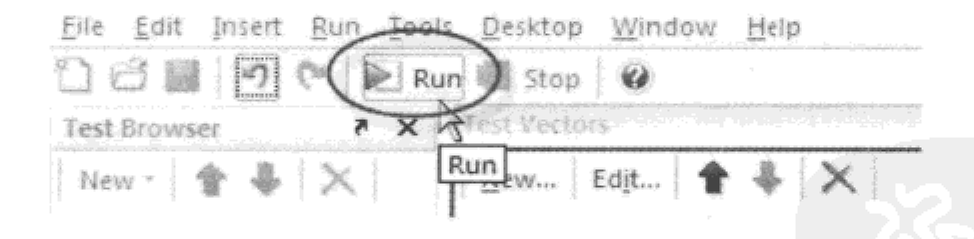


图 9.5.18　执行测试

SystemTest 主窗口右侧界面的 Run States 下部显示当前的迭代次数、测试时间，如图9.5.19 所示。

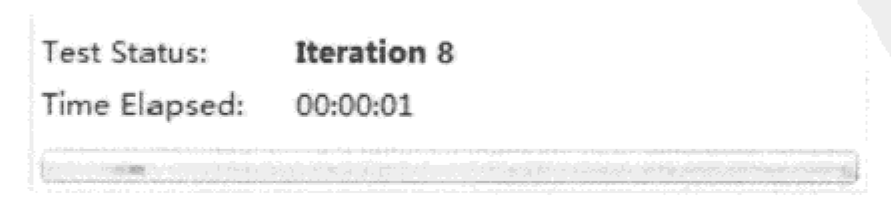


图 9.5.19　测试进度

完成后 Run States 界面给出测试结果文件链接与最终测试状态，本测试程序成功运行，如图 9.5.20 所示。

Generated Files

The following files were generated in D:\MATLAB FILES\DC_motor

Open	Filename
Test Results	outputs\test_results.mat

Final Test Status

Property	Value
Start Time	19-Oct-2010 11:14:42
Stop Time	19-Oct-2010 11:14:44
Iterations Completed	11
Final Status	Completed Successfully

Test Status: **Completed Successfully**

Time Elapsed: 00:00:02

图 9.5.20 测试完成

单击测试结果文件链接 Test Results，在 MATLAB 命令行窗口显示测试结果的详细信息，如图 9.5.21 所示。

```
Loading test results....
stresults =

     Test Results Object Summary for 'test':

            NumberOfIterations: 11
               TestVectorNames: Kf
              SavedResultNames:
                ResultsDataSet: [11x1 dataset]

     There are no Test Vector Groups associated with this test result object.

     Artifacts associated with this test result object:
       TEST-File (test.test)

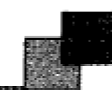

Type stresults.ResultsDataSet to display test results data. For information
on working with test results data, refer to the Analyzing Test Results demo.
```

图 9.5.21 测试结果信息

单击链接 stresults. ResultsDataSet，MATLAB 命令行窗口继续显示：

```
ans =

                 Kf           ToWorkspace
    I1      [      0]      [1524x1 double]
    I2      [0.1000]       [1523x1 double]
```



```
I3          [0.2000]     [1523x1 double]
I4          [0.3000]     [1522x1 double]
I5          [0.4000]     [1522x1 double]
I6          [0.5000]     [1522x1 double]
I7          [0.6000]     [1520x1 double]
I8          [0.7000]     [1520x1 double]
I9          [0.8000]     [1520x1 double]
I10         [0.9000]     [1520x1 double]
I11         [     1]     [1520x1 double]

>>
```

在 plot 区域中显示了取每个参数时，系统的响应曲线，如图 9.5.22 所示。其中纵坐标为信号幅度。横坐标为系统计算仿真时的采样序号，而不是时间，这是由于模型采用的是变步长求解器，每个采样点的间隔会有变化，不过每个采样点和相应的仿真时间点是一一对应的。这样就可以将该横坐标等效为时间坐标。

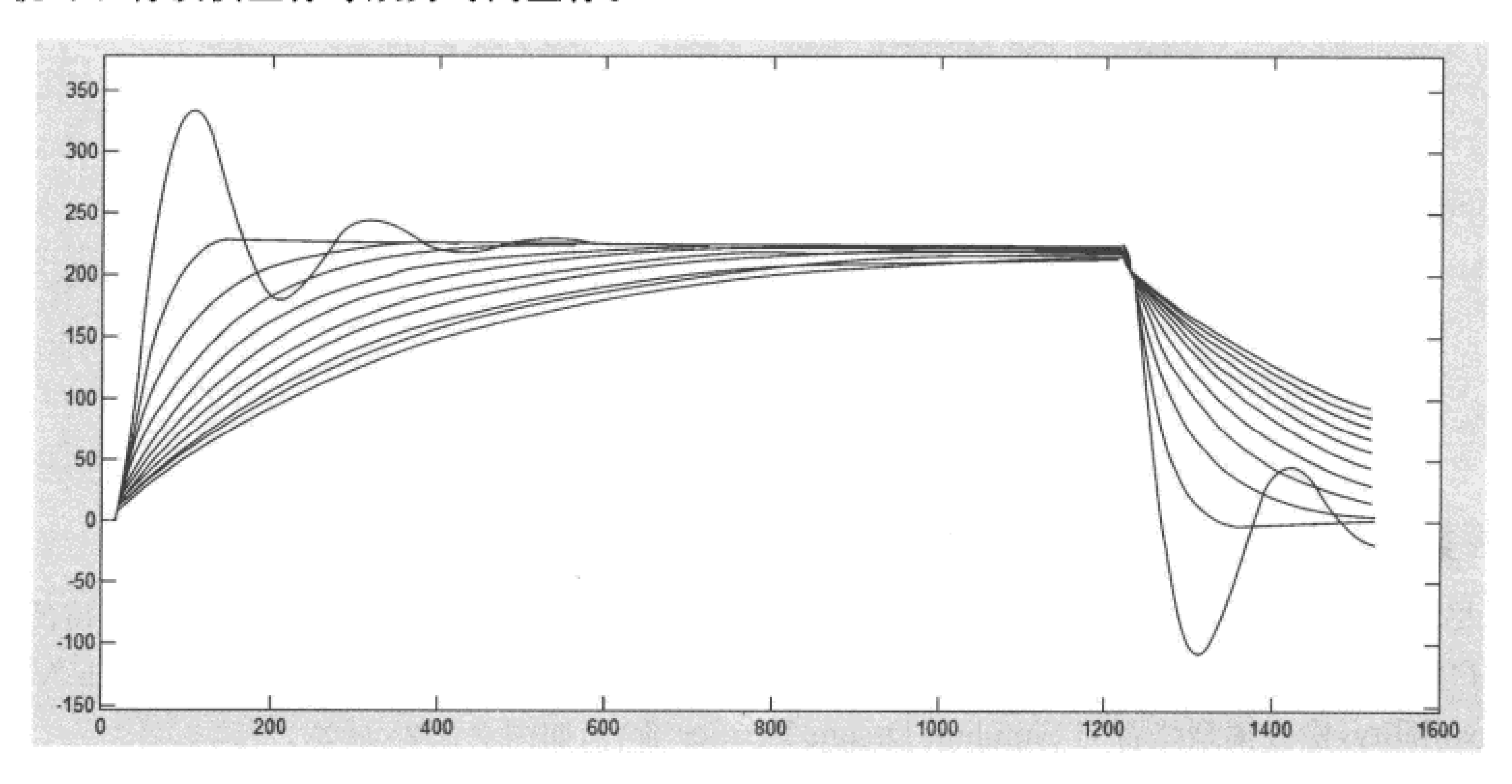


图 9.5.22 系统响应曲线

分析测试结果可知，电动机模型中的每个参数对整个系统的响应都有较大的影响，因此在对电动机建模时，应准确测试实际电动机的各项参数，否则用误差较大的参数得出的模型设计 PID 算法必定难以得到理想的效果。

9.5.2 Design Verifier

使用 Design Verifier 自动生成的测试用例，可达到满意的模型覆盖度以及用户自定义的目标，同时 Design Verifier 还可以验证模型的属性以及生成反例。

它支持以下几种模型覆盖度目标：分支覆盖度（decision）、条件覆盖度（condition）、变更条件/分支覆盖度（MC/DC）。当然用户也可以使用 design verification 模块，在 Simulink 或 Stateflow 模型里自定义测试目标。使用属性验证功能，用户可以发现设计的缺陷、遗漏的需求、多余的状态，这些问题在仿真过程中通常是很难发现的。

在 PID 控制直流电动机模型中，实现控制功能的部分只有 PID Controller 子系统，最终应用到嵌入式控制系统的代码也是由这部分模块生成。而电动机模型是为了模拟实际电动机的响应，因此，本节仅对 PID Controller 做 Design Verifier 测试用例分析。

在模型中仅保留 PID Controller 子系统，并为其添加输入、输出模块，如图 9.5.23 所示。

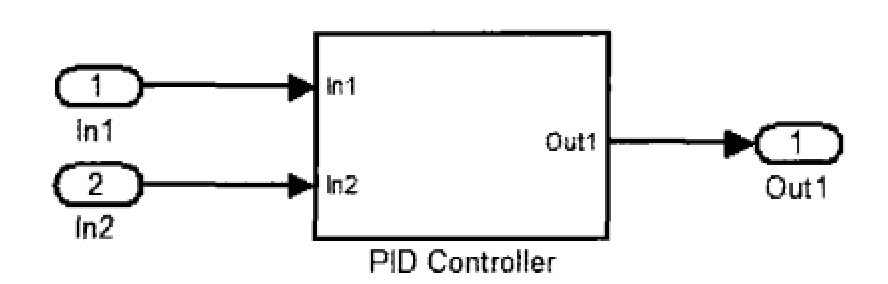


图 9.5.23　PID Controller 子系统

在 PID 模块设置中的 PID Advanced 选项卡中勾选 Limit Output 复选框，设置输出饱和。根据 PID Controller 的输出，可将上下限分别设为 1150 和－700，如图 9.5.24 所示。

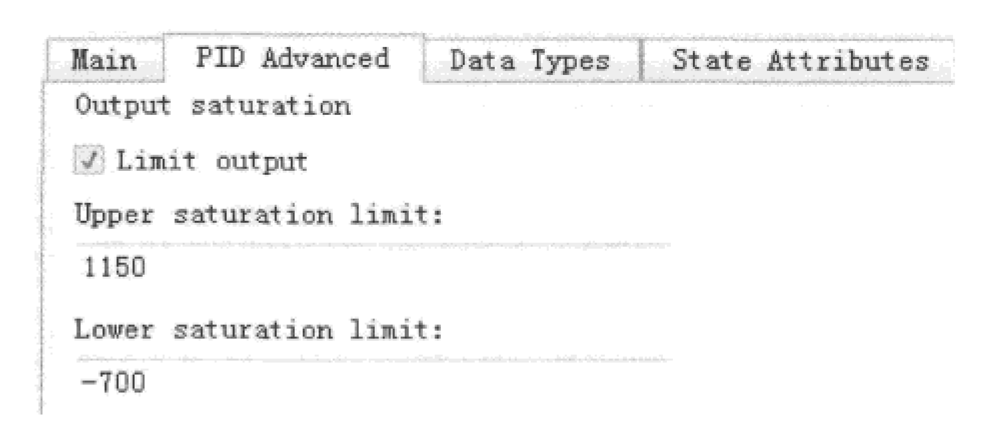


图 9.5.24　PID Advanced 选项卡

1. 兼容性检查

尽管 Simulink Design Verifier 支持许多 Simulink 与 Stateflow 特性，但仍有一些是不支持的，为此用户需要事先检查模型的兼容性。单击模型菜单项 Tools →Design Verifier → Check Model Compatibility，检查模型是否与 Simulink Design Verifier 兼容，如图 9.5.25 所示。

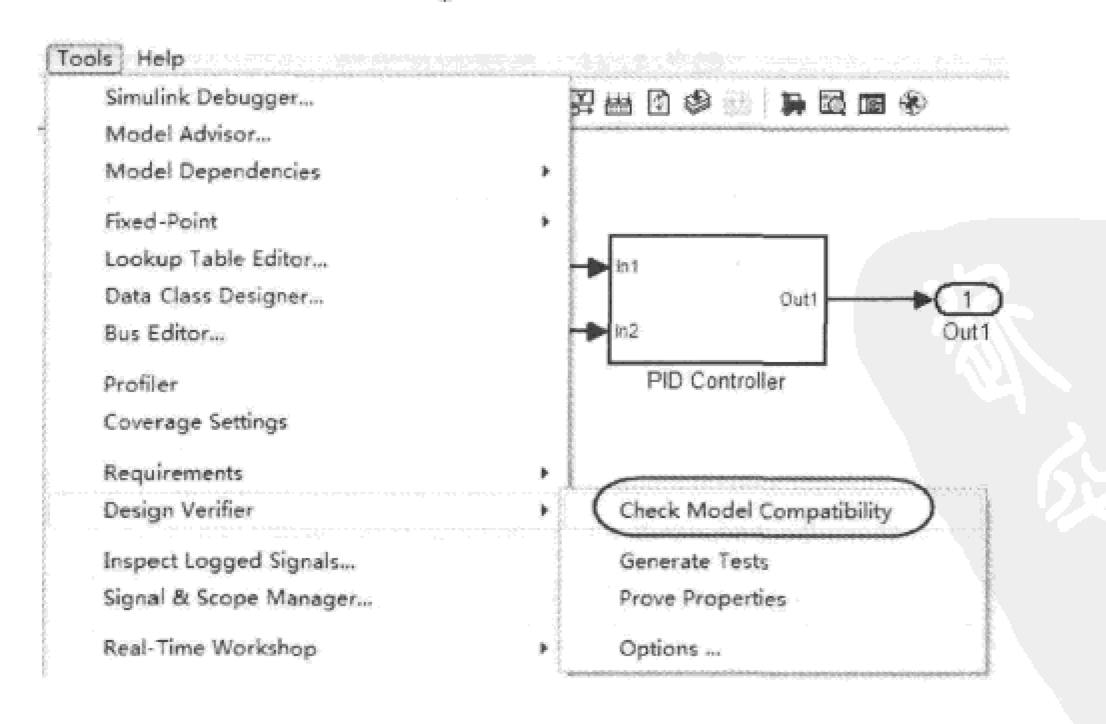
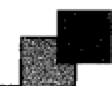


图 9.5.25　兼容性检查



由于 Design Verifier 仅支持定步长求解器，在日志中会提示不兼容的错误，如图 9.5.26 所示。

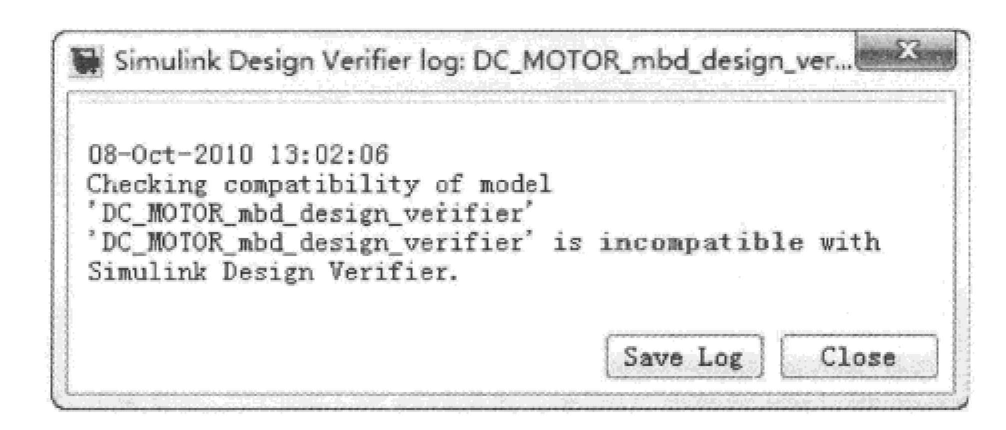


图 9.5.26　不兼容错误

选择菜单栏 Simulation→Configuration Parameters，在 solver options 界面选择定步长求解器，如图 9.5.27 所示。



图 9.5.27　求解器设置

这时在执行兼容性检查，则显示通过，如图 9.5.28 所示。

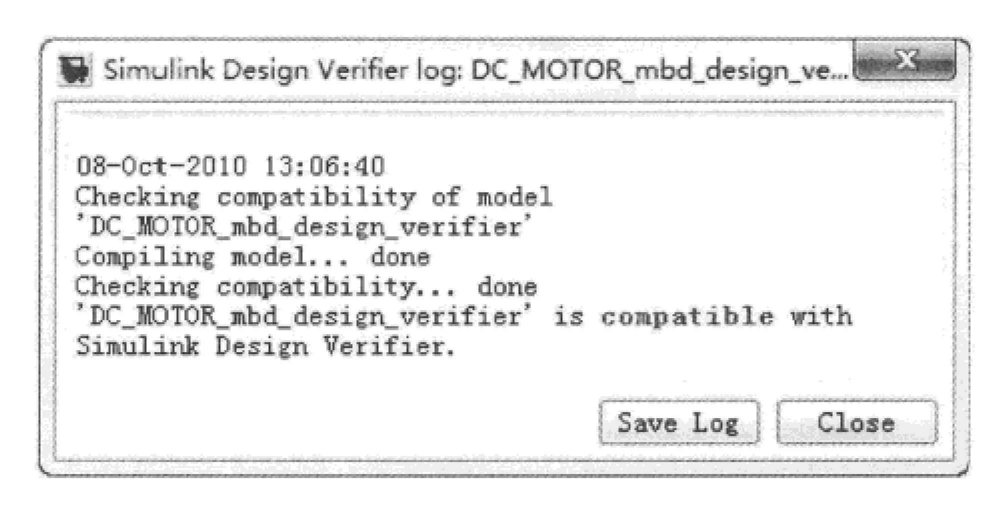


图 9.5.28　通过兼容性测试

2. 设置 Design Verifier 选项

单击模型菜单项 Tools →Design Verifier → Options...，模型参数配置窗口下部显示 Design Verifier 设置选项，如图 9.5.29 所示。

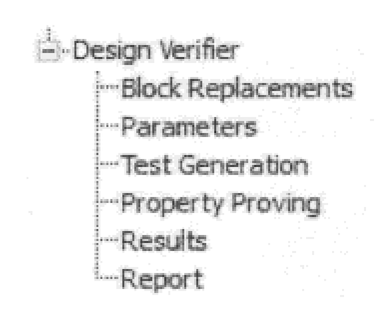


图 9.5.29　Design Verifier 设置选项

表 9.5.2 Design Verifier 设置项意义

选项卡	用　途
Design Verifier	指定 Design Verifier 的分析类型、时间、输出目录等
Block Replacements	指定用于模型预处理的模块替换规则
Parameters	指定参数设置
Test Generation	指定测试用例的生成类型
Property Proving	指定属性验证的选项
Results	指定输出的数据文件、测试用例模型、System Test 测试程序的生成选项
Report	指定测试报告的生成选项

本例在 Test Generation 选项卡界面设置 Test suite optimization 项目为 LongTestcases，这样模型的各种条件可在同一个测试用例里得到满足，便于模型与测试用例对照分析，如图 9.5.30所示。

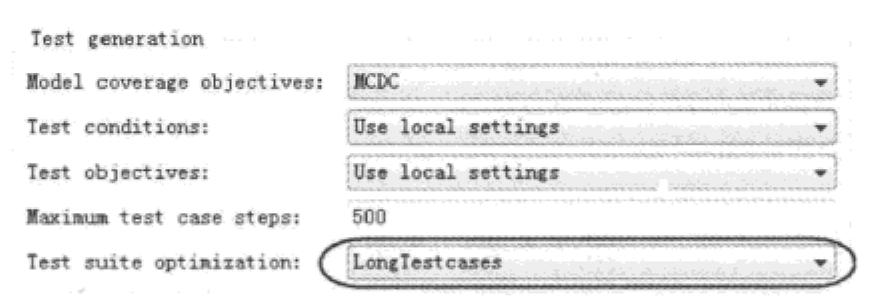


图 9.5.30 Test Generation 选项卡

3. 生成测试用例

单击模型菜单项 Tools → Design Verifier → Generate Tests，系统自动生成测试用例，如图 9.5.31 所示。

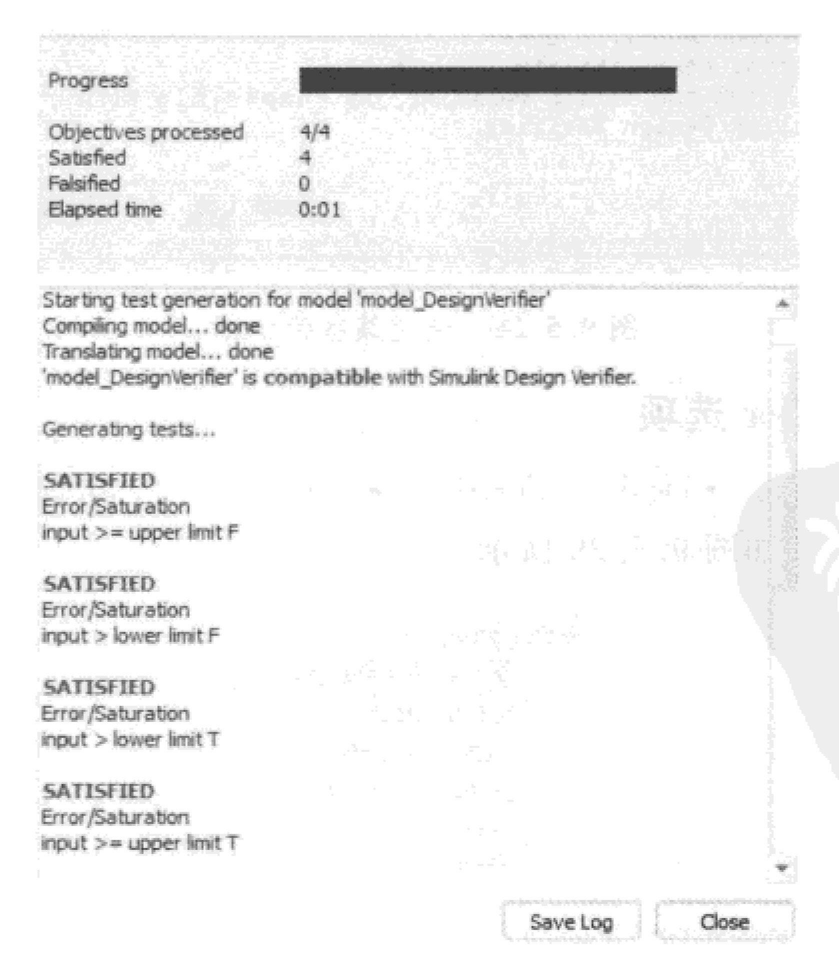


图 9.5.31 测试用例生成过程的日志



以下日志说明了测试用例生成的过程：

（1）生成测试用例：

```
Generating tests...


SATISFIED
PID Controller/PID Controller/Saturation
input >=  upper limit F
......
08-Oct-2010 13:15:01
Completed normally.
```

（2）生成测试用例对应的数据文件：

```
Generating output files:


  Data file:
  D:\MATLAB FILES\DC_motor\sldv_output\DC_MOTOR_mbd_design_verifier\DC_MOTOR_mbd_design_veri-
  fier_sldvdata.mat
```

（3）生成测试用例模型：

```
Harness model:
  D:\MATLAB FILES\DC_motor\sldv_output\DC_MOTOR_mbd_design_verifier\DC_MOTOR_mbd_design_veri-
  fier_harness.mdl
```

（4）生成 Simulink Design Verifier 报告：

```
Report:
  D:\MATLAB FILES\DC_motor\sldv_output\DC_MOTOR_mbd_design_verifier\DC_MOTOR_mbd_design_veri-
  fier_report.html

08-Oct-2010 13:15:05
Results generation completed.
```

4. 测试用例模型

该测试用例模型可用于分析模型覆盖度，如图 9.5.32 所示。

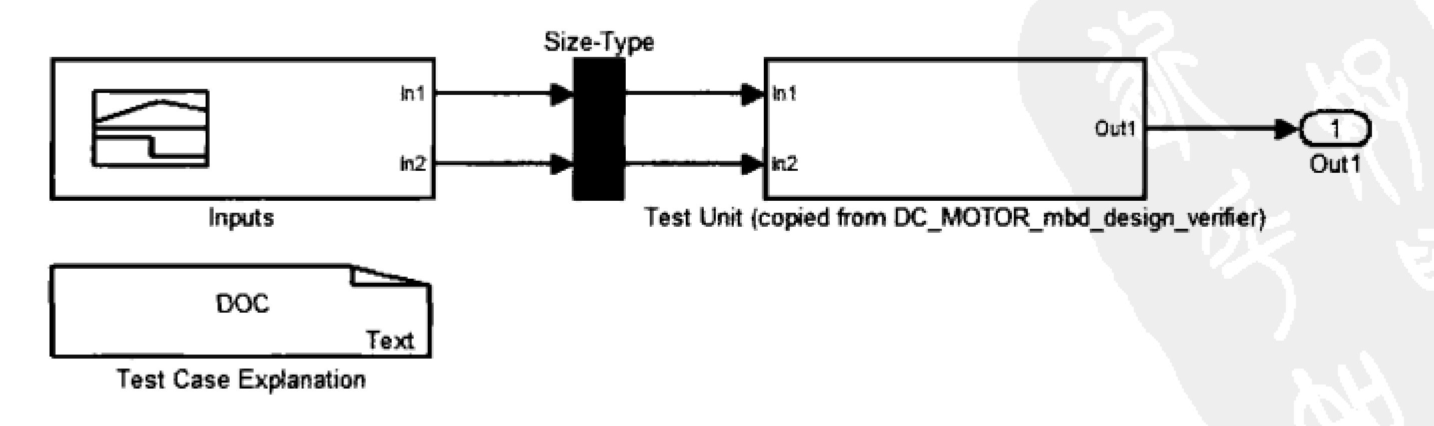


图 9.5.32　测试用例模型

双击模型左上角的 Input 模块，可打开 Signal Builder 窗口，如图 9.5.33 所示；双击模型右侧的 Test Unit，可查看原模型；双击模型左下角的 Test Case Explanation 模块，则在 MATLAB 编辑器窗口显示测试用例说明。

打开 Signal Builder 窗口与测试用例说明，可以更好地理解测试用例的作用。

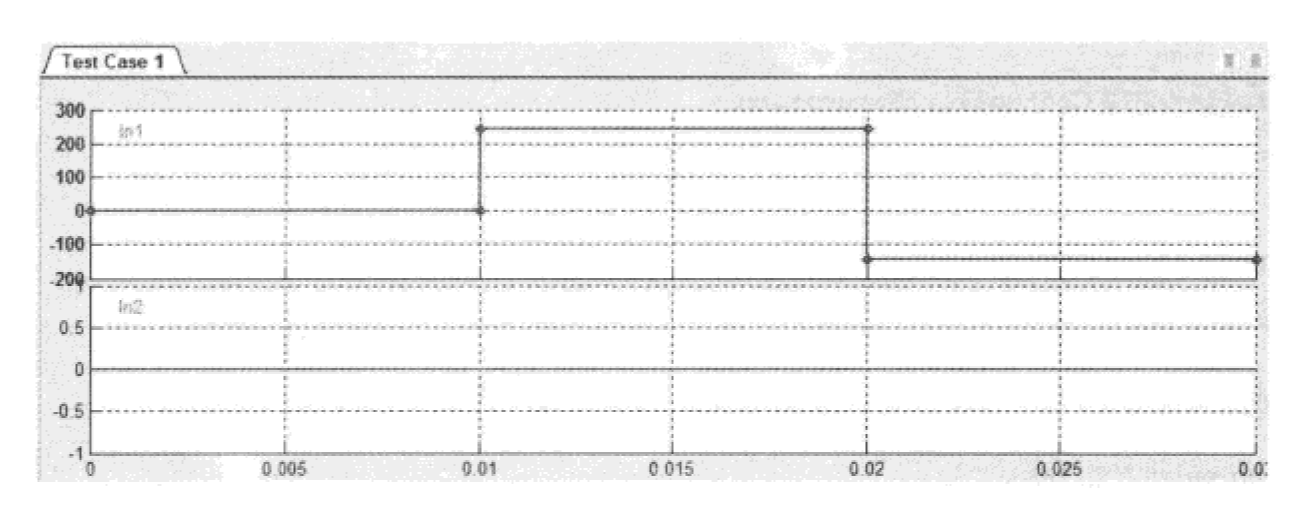


图 9.5.33　测试用例

```
Test Case 1 (4 Objectives)
    Parameter values:

    1. PID Controller/PID Controller/Saturation - input > lower limit F @ T = 0.02
    2. PID Controller/PID Controller/Saturation - input > lower limit T @ T = 0.00
    3. PID Controller/PID Controller/Saturation - input >= upper limit F @ T = 0.00
    4. PID Controller/PID Controller/Saturation - input >= upper limit T @ T = 0
```

5. Simulink Design Verifier 报告

Simulink Design Verifier 报告包括 5 个部分：

(1) 报告第 1 节 Summary 列出了 Design Verifier 分析的基本信息、测试目标的数量等，如图 9.5.34 所示。

Chapter 1. Summary

Analysis Information

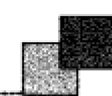
Model:	DC_MOTOR_mbd_design_verifier
Mode:	TestGeneration
Status:	Completed normally
Analysis Time:	0s

Objectives Status

Number of Objectives: 4
Objectives Satisfied:　4

图 9.5.34　第 1 节 Summary

(2) 报告第 2 节 Analysis Information，根据模型的不同，包含以下几个小节的全部或部分，如图 9.5.35 所示。



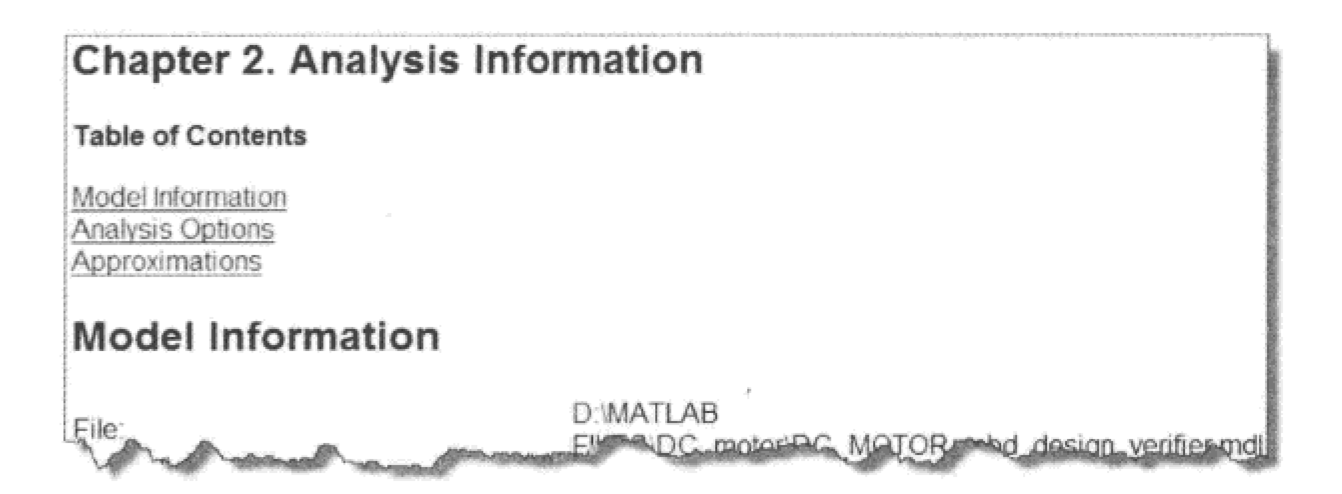


图 9.5.35　第 2 节 Analysis Information

表 9.5.3　报告第 2 节各小节意义

小　节	意　义
Model Information	列出了模型的基本信息：模型路径、修订版本、最后一次保存时间、作者等
Analysis Options	列出了 Design Verifier 的分析设置，选择菜单项 Tools → Design Verifier → Options... 可以修改这些设置
Unsupported elements	如果模型中包含不支持的元素，用户可以通过启用 automatic stubbing 功能，检查这些元素
Constraints	列出了 Design Verifier 软件在分析模型时的测试条件
Block Replacements Summary	如果 Design Verifier 软件替换了模型中的某些模块，则列出模块替换信息
Approximations	列出了 Design Verifier 软件在分析模型时使用的近似类型

(3) 报告第 3 节 Test Objectives Status 总结了整个模型的测试目标、目标类型、响应该测试的模块及描述，如图 9.5.36 所示。

Chapter 3. Test Objectives Status

Table of Contents

Objectives Satisfied

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

#	Type	Model Item	Description	Test Case
1	Decision	PID Controller/PID Controller/Saturation	input > lower limit F	1
2	Decision	PID Controller/PID Controller/Saturation	input > lower limit T	1
3	Decision	PID Controller/PID Controller/Saturation	input >= upper limit F	1
4	Decision	PID Controller/PID Controller/Saturation	input >= upper limit T	1

图 9.5.36　第 3 节 Test Objectives Status

(4) 第 4 节 Model Items 列出了每个被测模块的测试类型、描述、测试状态等，如图9.5.37 所示。

Chapter 4. Model Items

Table of Contents

PID Controller/PID Controller/Saturation

This section presents, for each object in the model defining coverage objectives, the list of objectives and their individual status at the end of the analysis. It should match the coverage report obtained from running the generated test suite on the model, either from the harness model or by using the sldvruntests command.

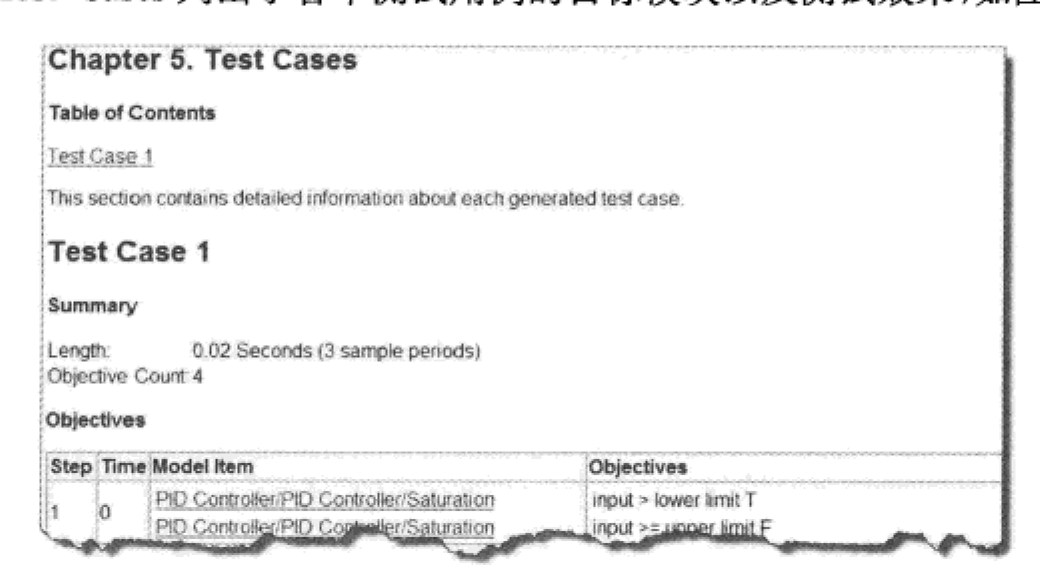
PID Controller/PID Controller/Saturation

View

#:	Type	Description	Status	Test Case
1	Decision	input > lower limit F	Satisfied	1
2	Decision	input > lower limit T	Satisfied	1
3	Decision	input >= upper limit F	Satisfied	1
4	Decision	input >= upper limit T	Satisfied	1

图 9.5.37　第 4 节 Model Items

(5) 第 5 节 Test Cases 列出了各个测试用例的目标模块以及测试效果，如图 9.5.38 所示。



Chapter 5. Test Cases

Table of Contents

Test Case 1

This section contains detailed information about each generated test case.

Test Case 1

Summary

Length:　0.02 Seconds (3 sample periods)
Objective Count: 4

Objectives

Step	Time	Model Item	Objectives
1	0	PID Controller/PID Controller/Saturation	input > lower limit T
		PID Controller/PID Controller/Saturation	input >= upper limit F

图 9.5.38　第 5 节 Test Cases

6. 覆盖度分析报告选项

在分析之前，用户应指定必要的模型覆盖度选项，选择模型窗口的菜单项 Tools →Coverage Settings...，打开覆盖度设置对话框，如图 9.5.39 所示。

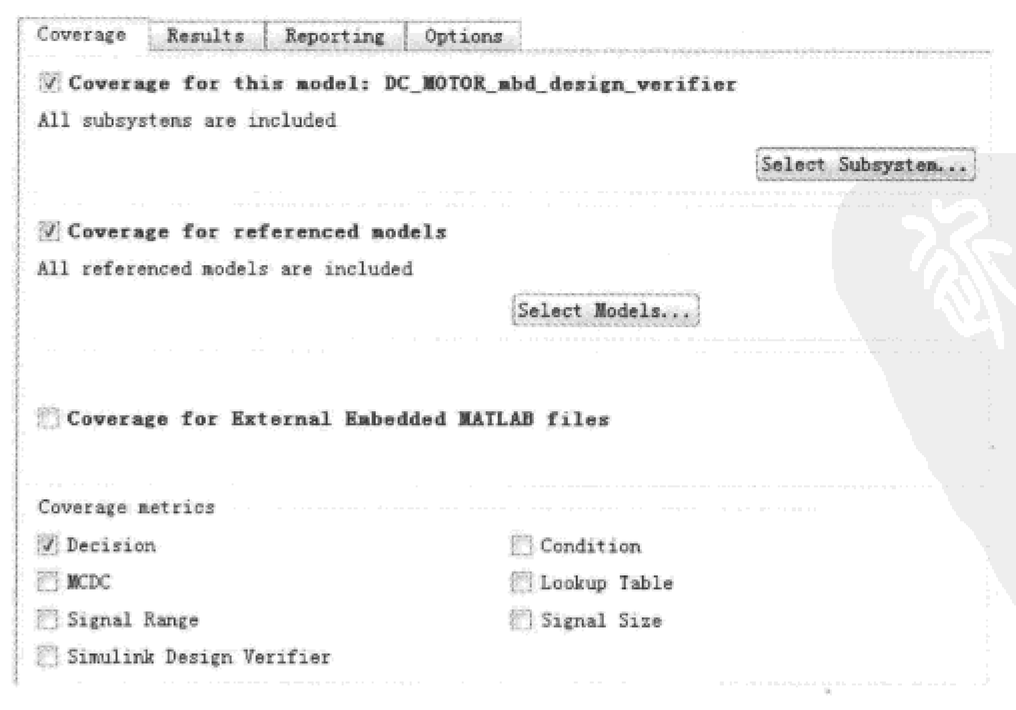
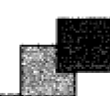


图 9.5.39　覆盖度设置对话框



(1) Coverage 选项卡。

① 选择模型及子系统。勾选 Coverage for this model 前的复选框或继续单击 Select Subsystem... 按钮,选择需要检查的子系统。仿真过程中,系统将收集并报告选定模型或子系统的覆盖度信息。

② 选择引用模型。对于包含引用模块的模型,可勾选 Coverage for referenced models 复选框或继续单击 Select Models... 按钮,选择需要分析的引用模型。仿真过程中,系统将收集并报告全部或个别指定的引用模型的覆盖度信息。

Simulink V&V 软件,仅针对工作于 Normal 仿真模式下的引用模型给出覆盖度报告,对于 Accelerator 仿真模式,Simulink V&V 软件是无法记录其覆盖度的。

③ 检查外部 Embedded MATLAB 文件。勾选 Coverage For External Embedded MATLAB Files 复选框,在仿真过程中,系统将收集并报告模型中 Embedded MATLAB Function 模块或 Stateflow 图表中调用到的 M 文件的覆盖度信息。

④ 选择覆盖度检查类型。模型覆盖度分析类型如表 9.5.4 所列。

表 9.5.4 覆盖度分析类型

类 型	名 称
Cyclomatic Complexity	循环复杂度
Decision Coverage (DC)	判决覆盖度
Condition Coverage (CC)	条件覆盖度
Modified Condition/Decision Coverage (MCDC)	修改的判决/条件覆盖度
Lookup Table Coverage	查表覆盖度
Signal Range Coverage	单一范围覆盖度
Signal Size Coverage	信号大小覆盖度
Simulink Design Verifier Coverage	Simulink 设计验证覆盖度

有些 Simulink 模块可以接受任何一种的覆盖度检查,而有些模块却只能接受其中的某些检查。对于 Stateflow 状态、事件及状态时序逻辑判决,只能得到判决覆盖度;对于 Stateflow 状态转移,则可以得到 DC、CC、MCDC 三种覆盖度。对于上述参数设置及覆盖度类型的意义及各模块可接受的检查类型,用户应事先参阅帮助文档。

对于不同的引用模型,如需要各自执行不同类型的覆盖度检查,可以使用命令 cv.cvtestgroup 与 cvsimref,具体用法请参阅帮助文档。

(2) Results 选项卡。该页有 5 个选项,如图 9.5.40 所示。

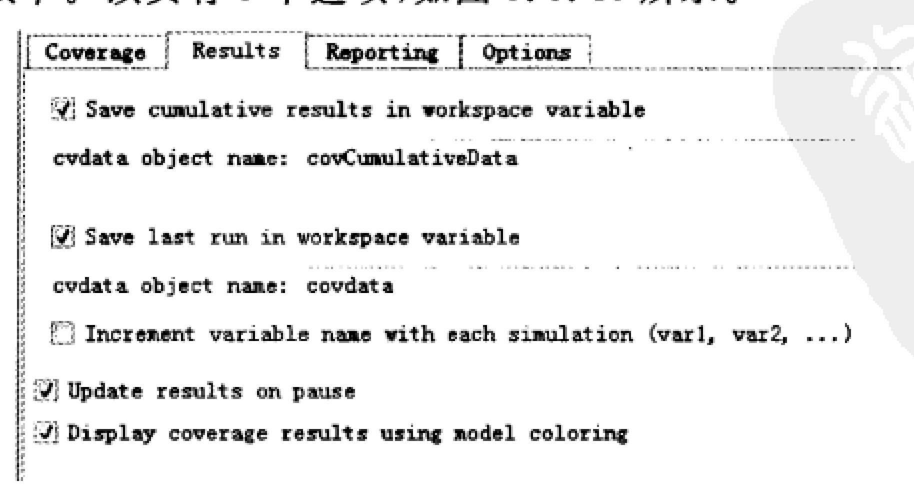


图 9.5.40 Results 选项卡

① Save Cumulative Results in Workspace Variable。累积各次的覆盖度检查结果，以 cvdata object name 文本框内的文字作为对象变量名，保存在 MATLAB 工作空间。

② Save Last Run in Workspace Variable。在 MATLAB 工作空间保存最近一次的覆盖度检查结果，以 cvdata object name 文本框内的文字，作为对象变量名。

③ Increment Variable Name with Each Simulation。按顺序逐个保存最近一次的覆盖度检查结果，避免以往的结果被覆盖。

④ Update Results on Pause。系统在仿真暂停时，立即给出覆盖度检查报告，随后用户可以继续执行仿真，在下一次暂停或停止时，再次更新检查报告。

⑤ Display Coverage Results Using Model Coloring。根据覆盖度程度，以不同颜色显示各个模块。例如全覆盖的模块，显示淡绿色，不完全覆盖的模块，显示淡红色。

(3) Report 选项卡。该选项卡各选项，用于设置是否生成以及如何生成检查报告，如图 9.5.41所示。

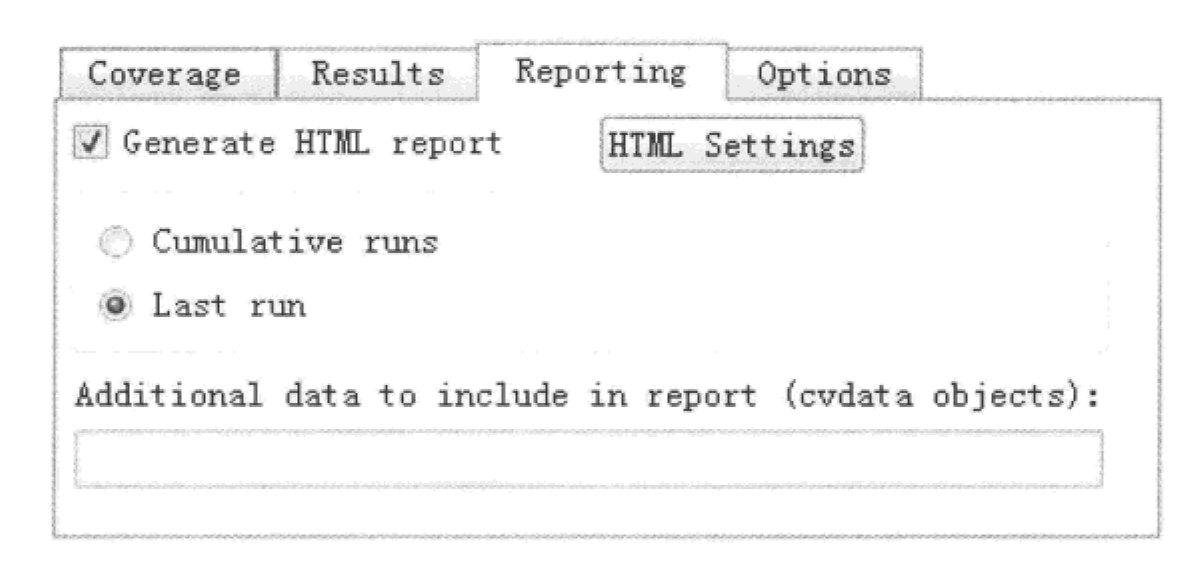


图 9.5.41 Report 选项卡

① Generate HTML Report。当系统完成覆盖度检查时，生成检查报告，同时在 MATLAB 网页浏览器显示，如图 9.5.42 所示。

② 单击右侧的 Settings 按钮，用于设置报告选项，意义如表 9.5.5 所列。

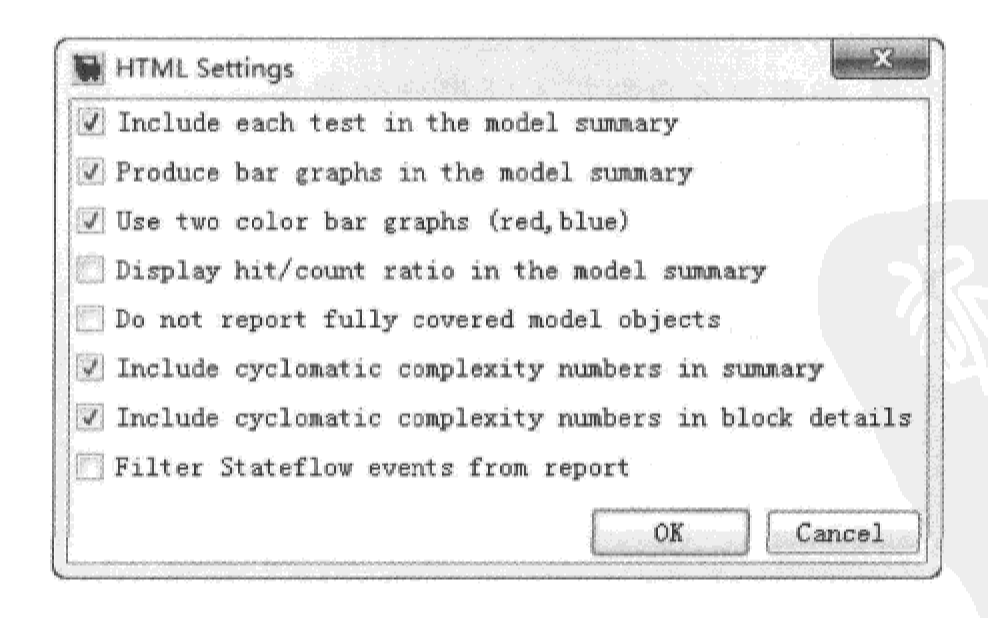


图 9.5.42 报告选项

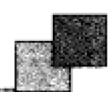


表 9.5.5　模型覆盖度报告选项

报告选项	意　义
Include each test in the model summary	在报告上部，给出每一个测试用例的覆盖度列表，如果取消该项，则仅给出总的覆盖度报告
Produce bar graphs in the model summary	以条状图例显示覆盖度
Use two color bar graphs (red, blue)	用红蓝两色替代条状图例中的黑白两色，便于识别
Display hit/count ratio in the model summary	以百分比及分数形式同时显示覆盖度，如 80% (4/5)
Do not report fully covered model objects	报告里不体现全覆盖模块，这样能明显减小报告，这在测试用例开发阶段特别有用
Include cyclomatic complexity numbers in summary	在报告里体现模型及其第一级子系统、Stateflow 图表的循环复杂度。在计算复杂度时，系统将子系统以及 Stateflow 图表看作一个对象，以黑体显示复杂度数值
Include cyclomatic complexity numbers in block details	在报告的模块细节部分体现循环复杂度检查结果
Filter Stateflow events from report	排除 Stateflow 事件的覆盖度数据

③ Cumulative Runs。本次生成的报告里体现此前各次积累的覆盖度检查结果。

④ Last run。本次生成的报告里仅体现最近一次的检查结果。

⑤ Additional data to include in report。本次生成的报告里加入以前检查的结果，在文本框区域输入数据的名称。

(4) Options 选项卡。

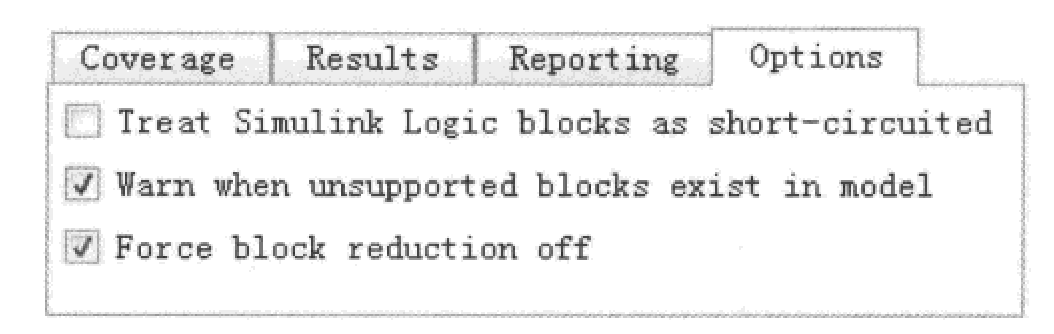


图 9.5.43　Options 选项卡

① Treat Simulink Logic blocks as short-circuited。在进行覆盖度检查时，对于逻辑模块，如果某一个输入能立即决定该模块的输出，则忽略该模块的其他输入。例如逻辑与模块，如果有一个输入“非”，则整个模块输出“非”，其他输入不再起作用，如同将“非”输入端与输出端短路。

该选项仅对 DC 及 MCDC 覆盖度检查有效，这可能会导致模型覆盖度达不到 100%。

② Warn when unsupported blocks exist in model。如果模型里含有无法执行覆盖度检查的模块，在仿真结束时，将提出警告信息。

③ Force block reduction off。如果用户在模型参数配置窗口启用了 Block reduction，如果勾选该复选框，则系统在收集模型覆盖度信息时，忽略 Block reduction 选项。

7. 模型覆盖度报告

完成上述设置后，选中 Signal Builder 窗口所示的某个测试用例，单击工具栏上的仿真按钮，即得到该次测试的覆盖度报告，如图 9.5.44 所示。

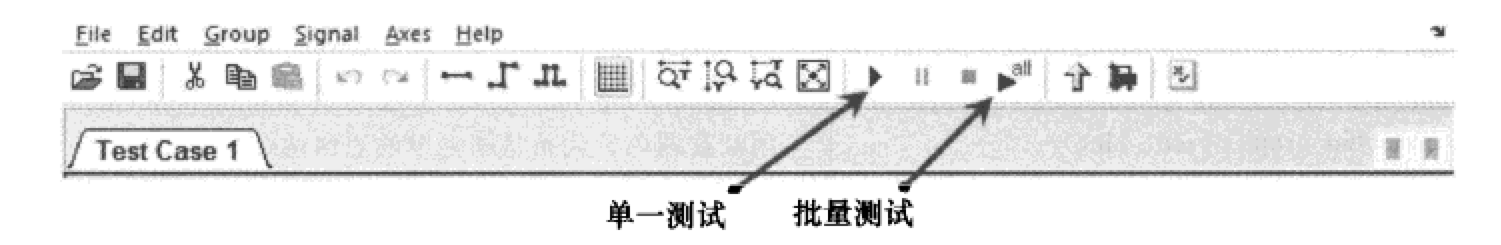


图 9.5.44 运行测试用例

报告显示可以得到 100％的模型覆盖度，如图 9.5.45 所示。

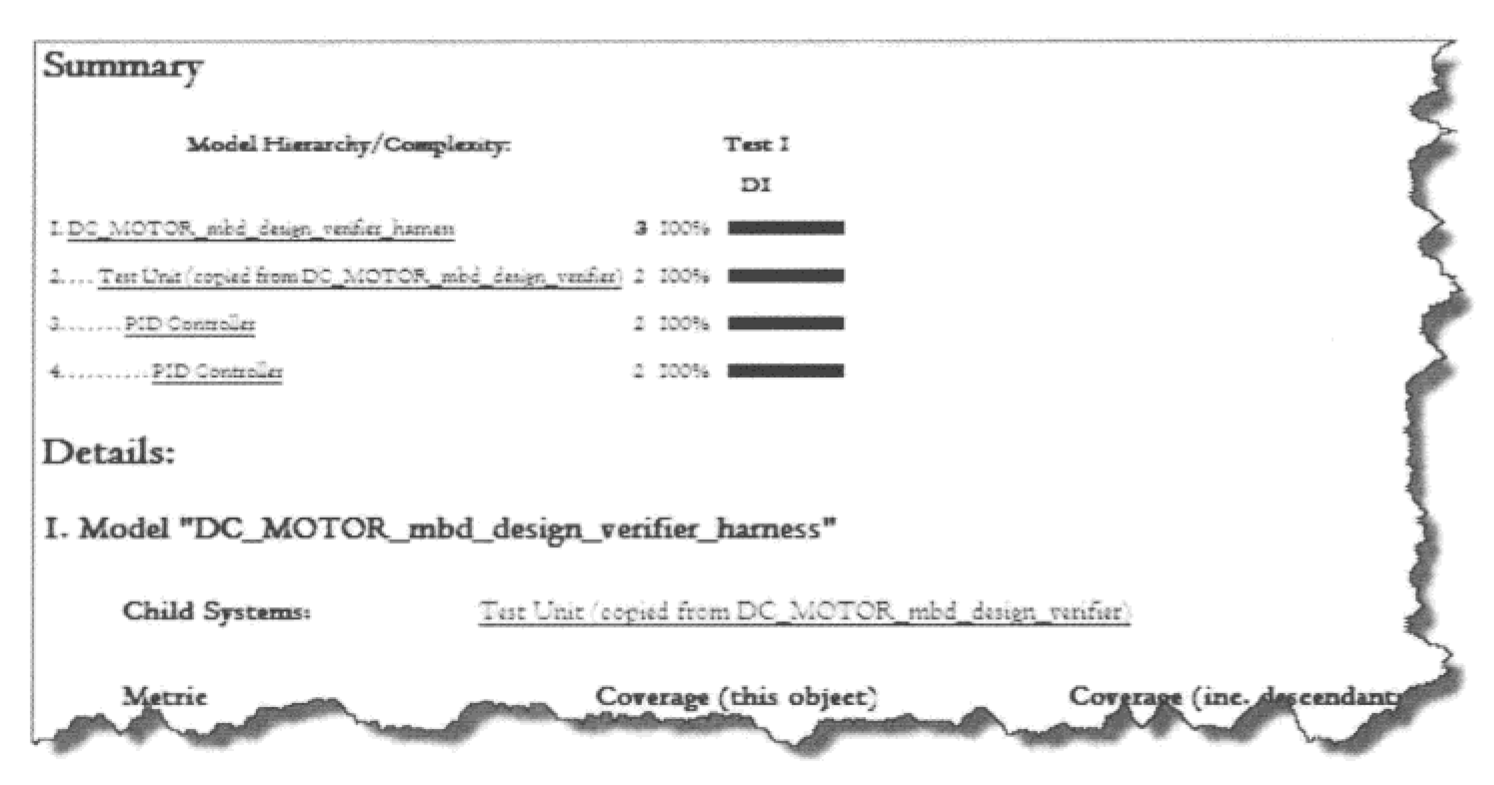


图 9.5.45 覆盖度报告

Design Verifier 主要是为了测试含有复杂逻辑的多分支系统，例如含有 if... else... 等结构时，Design Verfier 生成测试用例，测试系统在任意可能情况下的响应是否符合要求。而本例并不含有这些结构，测试用例仅仅针对输出饱和的上下限，使这项测试显得意义不大。但本例仍不失为一个如何使用 Design Verifier 工具的例程。

9.5.3 Model Advisor 检查

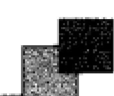
1. 启动 Model Advisor

在启动 Model Advisor 前，要确保当前目录未写保护。启动之后，Model Advisor 在当前目录下建立一个子目录 slprj，用于存放检查报告及其他信息。

用户可以在模型窗口，选择菜单项 Tools→Model Advisor，在 System Selector 窗口(图9.5.46)选择总体模型或模型下的子系统。

对于子系统，用户可以直接选择子系统右键菜单的 Model Advisor 命令。

本演示过程选择总体模型 DC_MOTOR_mbd。



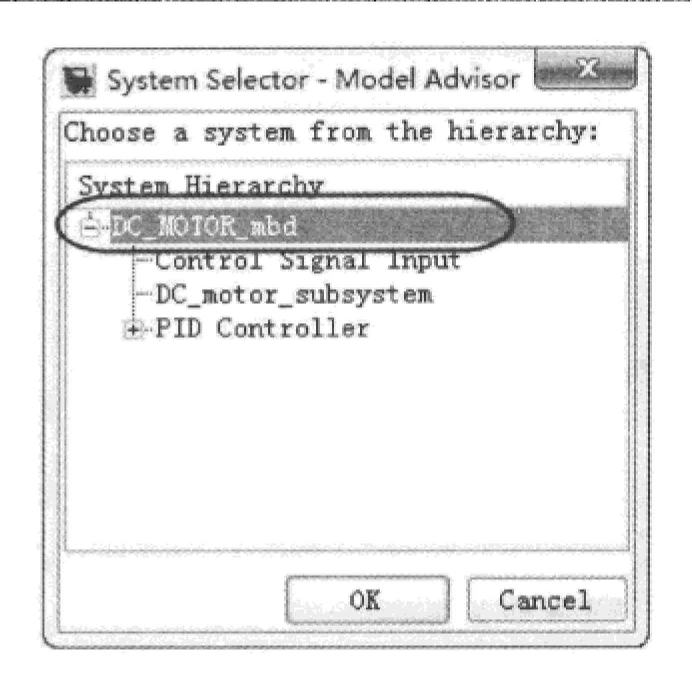


图 9.5.46　启动 Model Advisor

2. Model Advisor 窗口

选择模型或子系统后，系统显示 Model Advisor 窗口（图 9.5.47）。

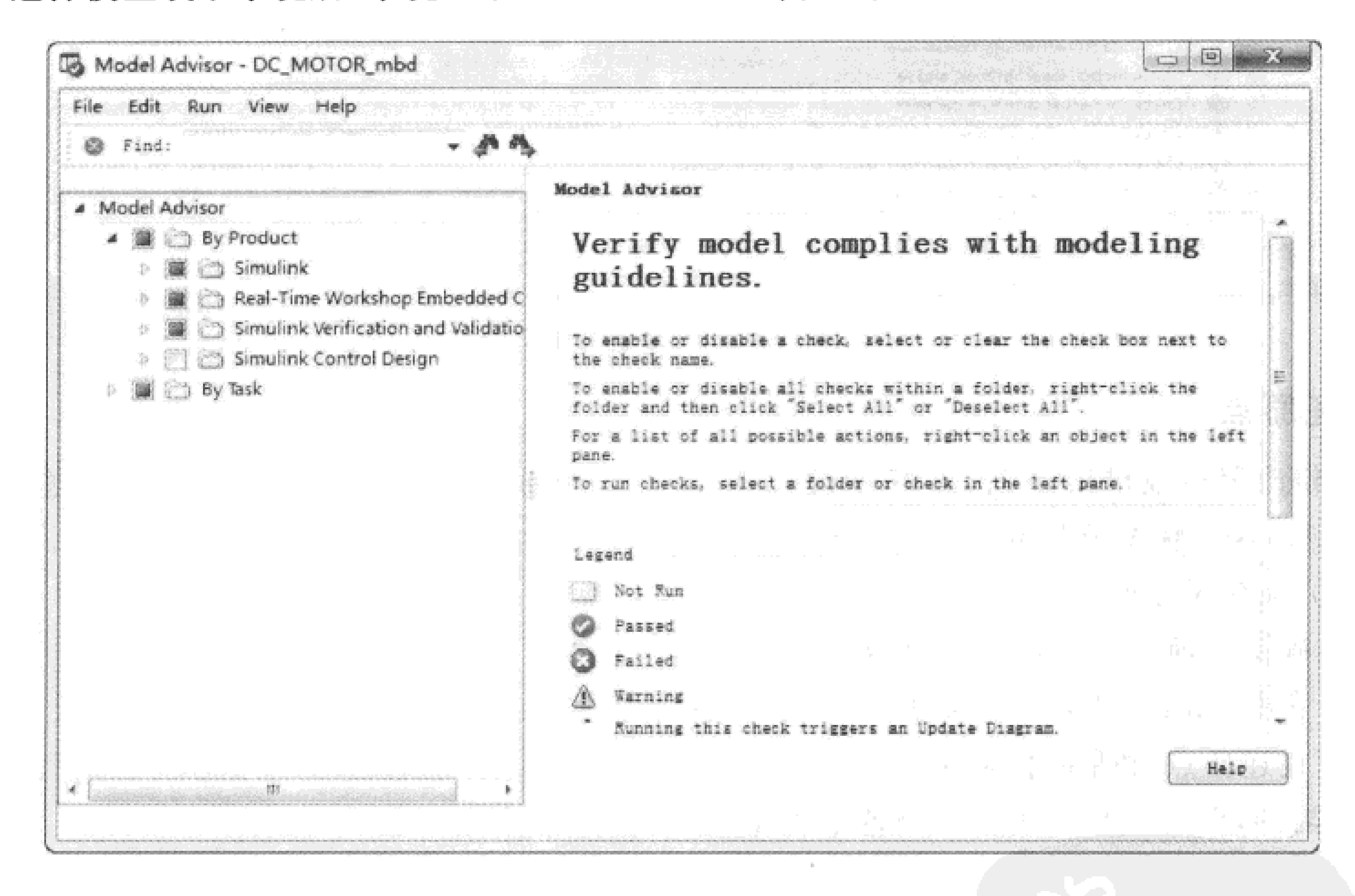


图 9.5.47　Model Advisor 窗口

展开窗口左侧的任务树，选中某一类或某一项检查，右侧任务管理窗口显示对应项目的检查说明、检查结果、检查按钮等。

如果用户先前已经执行了模型检查，那么下一次打开 Model Advisor 时，则显示上一次的检查结果、修改建议等，再次执行检查后，旧的检查结果将被覆盖。

选择菜单 Edit→reset 可以将撤销当前的所有检查结果。

3. 执行检查

(1) 展开节点 By Product，勾选下属的 Simulink 检查项前的复选框。

(2) 单击节点 Simulink,单击右侧页面的 Run Selected Checks 按钮,开始检查。

(3) 检查完成,右侧窗口显示检查结果(图 9.5.48)。✔表示通过检查项目,✖表示失败项目,⚠表示警告项目,☐表示未检查项目;左侧任务树的节点图标对应显示了该项检查的状态。

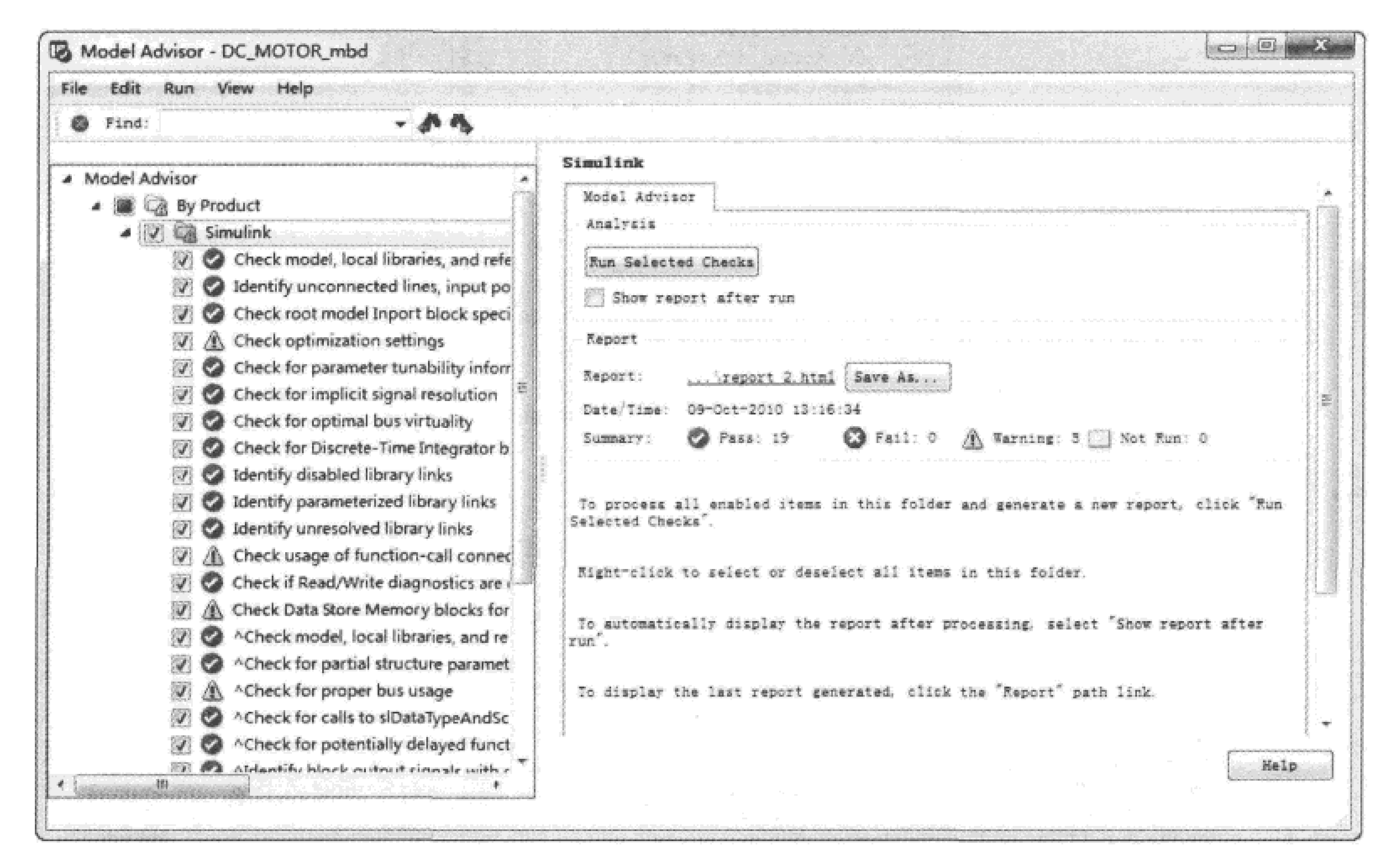


图 9.5.48 检查结果

4. 还原点

在修改警告或错误之前,最好先设置还原点,以便撤销操作。还原点只保存当前模型、Model Advisor 状态及基本工作空间变量,不保存诸如库文件、引用子模型等其他信息。

(1) 保存还原点。

① 确保当前的工作目录为模型所在的目录。

② 选择 Model Advisor 窗口菜单项 File→Save Restore Point As...。

③ 在还原点对话框下部的 Name、Description 栏输入还原点名称及描述文字,虽然描述不是必需的,但可以帮助用户区分各个还原点(图 9.5.49)。

④ 单击 Save 按钮,保存还原点。

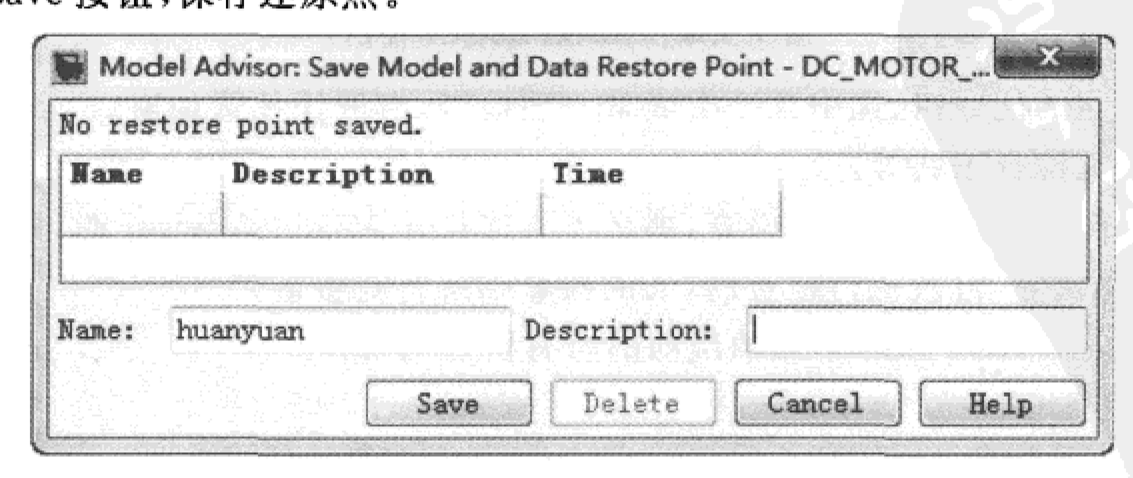
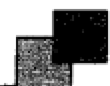


图 9.5.49 保存还原点



用户也可以选择 Model Advisor 窗口菜单项 File→Save Restore Point，快速保存还原点，这时系统使用默认名称 autosaven，其中 n 为顺序号。使用这样的保存方式，无法后期修改还原点名称及描述。

(2) 导入还原点。

① 选择 Model Advisor 窗口菜单项 File→Load Restore Point。

② 对话框列出所有的还原点名称、描述、保存时间，用户可据此选择需要的还原点(图9.5.50)。

③ 单击 Load 按钮，在后续出现的确认对话框中继续单击 Load 按钮，完成还原。

用户还可以选中已有的还原点，单击 Delete 按钮，删除还原点。

Model Advisor: Load Model and Data Restore Point - DC...

Select a restore point to load.

Name	Description	Time
huanyuan		09-Oct-2010 13:34:11

Load　Delete　Cancel　Help

图 9.5.50　导入还原点

5. 修改警告及错误

警告及错误说明当前模型或子系统的设置并不是最优的，用户可以根据检查报告修改对应的警告及错误。

本节只说明修改的一般过程，不具体讨论某一项检查的意义，各类检查的详细说明，可以在帮助窗口输入 Simulink Checks、Simulink Verification and Validation Checks 等关键字查找，用户可以根据实际需要确定合适的检查项目。

修改警告及错误的方式有以下 3 种：

(1) 手动修改。对于每一个未通过的检查项目，Model Advisor 都给出了修改建议，例如检查项 Check optimization settings 标记了黄色惊叹号⚠，单击该节点，右侧界面 Analysis 结论区显示修改建议(图 9.5.51)。

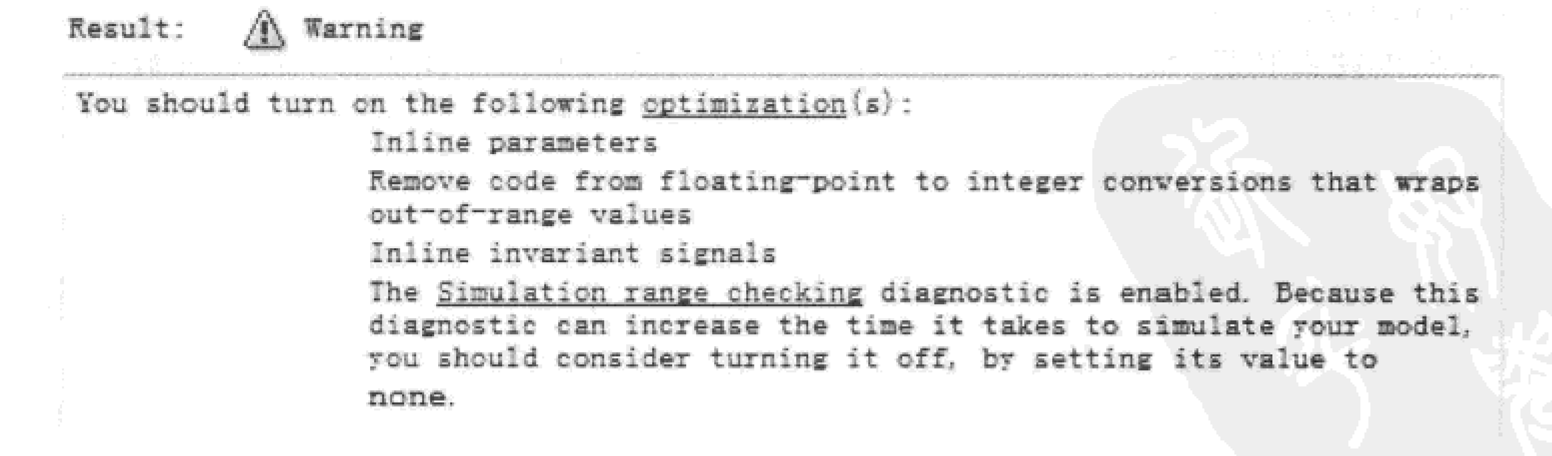


图 9.5.51　手动修改

① 分别单击链接 optimization 和 Simulation range checking，系统自动打开模型配置对话

框，按修改建议勾选对应复选框。

② 单击Run This Check，再次检查该项目，节点图标变成✔，说明检查通过。

(2) 自动修改。有些检查项提供了自动修改功能，它能自动执行 Analysis 结论区列出的所有修改意见。因此用户需要事先了解这些修改意见的意义，如果不需要全部修改，则可不使用自动修改功能。例如检查项 Check usage of function-call connections 标记了黄色惊叹号⚠，单击该节点，再单击右侧页面下部 Action 区的按钮Modify Settings，结果区列出本次修改的内容，如图 9.5.52 所示。

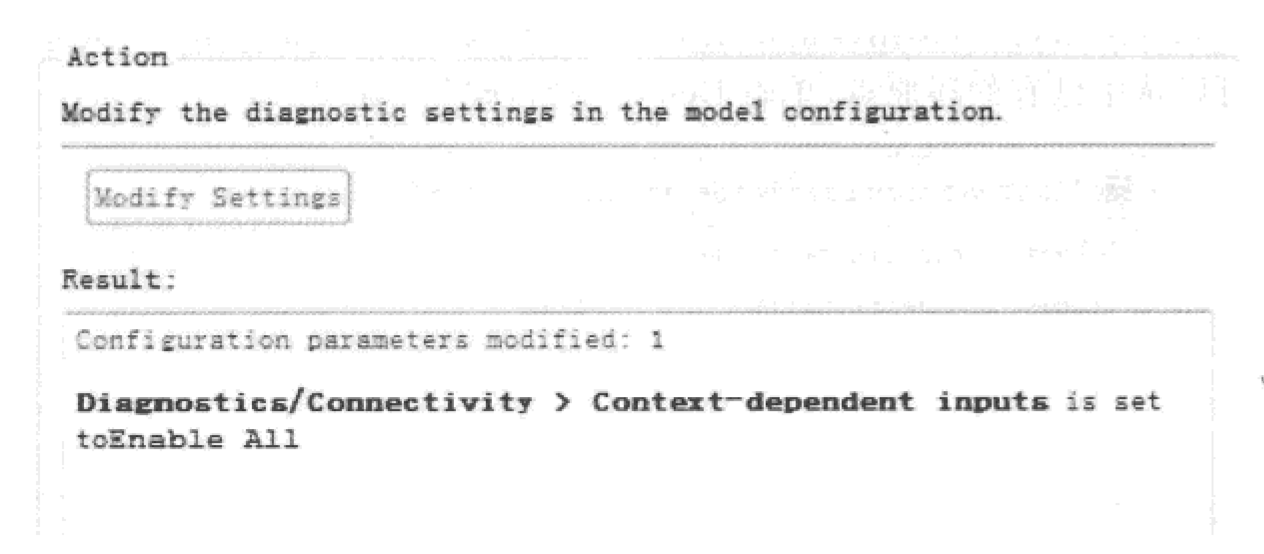


图 9.5.52 自动修改

(3) 批量修改。对于有些检查项，Model Advisor 提供了批量修改功能，Explore Result 按钮处于有效状态，单击该按钮用户可以在一个类似模型浏览器的窗口里快速定位、修改模型及模块参数，避免逐个模块修改，也避免错误修改原本正确的设置。

9.6 定点模型

进行定点信号处理能简化电路，可以使得运算简单、运算速度提高，减少运算时间，由于定点信号处理有这些优点，所以定点信号处理应用较广。对于嵌入式系统要求低功耗，而且芯片的体积小，这些要求定点信号处理都满足，而浮点模型功耗大，一般都将浮点模型转成定点模型进行信号处理。

而定点算法中数据类型是有限字长的，容易引入量化误差，产生溢出，所以在将设计的数据类型从浮点转化为定点需要考虑以下三点：

(1) 根据硬件特性、输入、输出及中间信号的动态范围来选择合适的字长和分数长度。

(2) 在定点运算中限制及降低量化误差的传播。

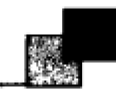
(3) 在硬件实现前保证定点设计满足需求。

有两种方法可以将浮点模型转成定点模型进行信号处理：

(1) 手动设定字长，再借助 Fixed-point Tool 工具检查设置是否满足设计要求，如果在运算过程中变量发生溢出，则用 Fixed-point Tool 工具自动定标。

(2) 利用 Fixed-Point Advisor 工具自动定标，再借助 Fixed-point Tool 工具进一步优化定标。

Fixed-Point Advisor 工具可将模型从浮点转换成定点，产生初始定标来调整模型配置参数和模块配置参数，为模块和数据配置初始定标，针对浮点模型来验证初始定标结果，并为代码生成做准备。



Fixed-Point Tool 工具可以进一步优化定标，它只对模型中已经设置为定点数的变量进行分析和操作，并可以为嵌入式代码优化定标，折衷范围和精度，使数据不溢出，产生优化的定标。而且允许用户指定仿真或设计的数据类型及记录模式(最小、最大值和溢出)，选择自动定标信号或锁定它们以防止改变，使用数据类型覆盖来支持浮点到定点表示的快速切换，还可产生浮点设计与定点设计的差值比较图。

9.6.1　Fixed Point Advisor

本节将用 PID 控制电动机模型的例子具体说明如何使用 Fixed Ponit 工具对模型定点化。如前文所述，最终的嵌入式实现代码仅与 PID Controller 子系统有关，因此这里也仅对该模块作定点化处理。

由于模型中的 DC_motor_subsystem 中含有不支持离散求解器的模块，首先应将菜单项 Simulation→Configuration Parameters 中 Solver 界面的求解器设置为可变步长，如图 9.6.1 所示。

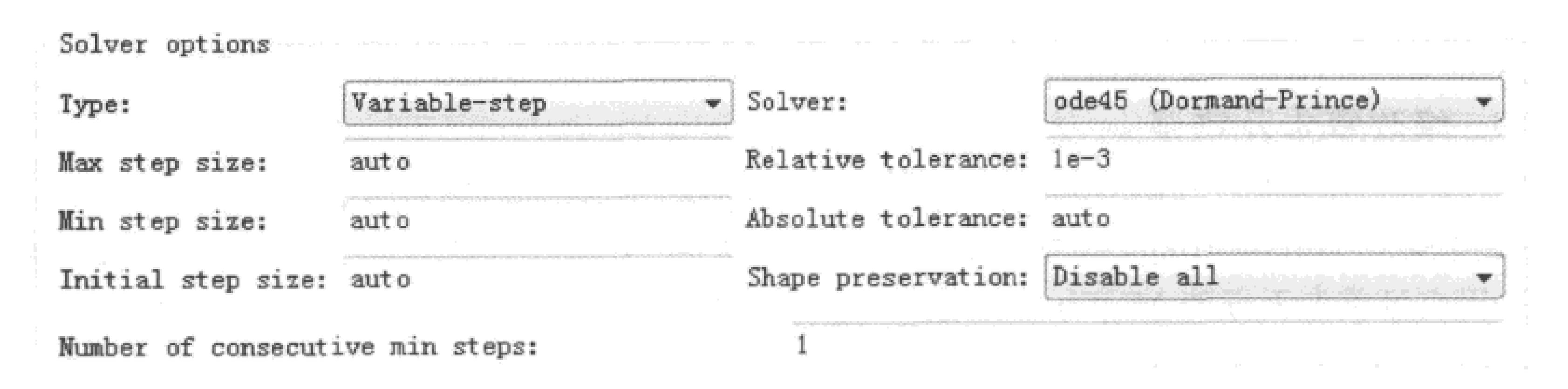


图 9.6.1　求解器设置

1. 启动 Fixed Point Advisor

在 PID Controller 子系统上右击，选择 Fixed Point→Fixed Point Advisor 命令，打开图9.6.2 所示的对话框。

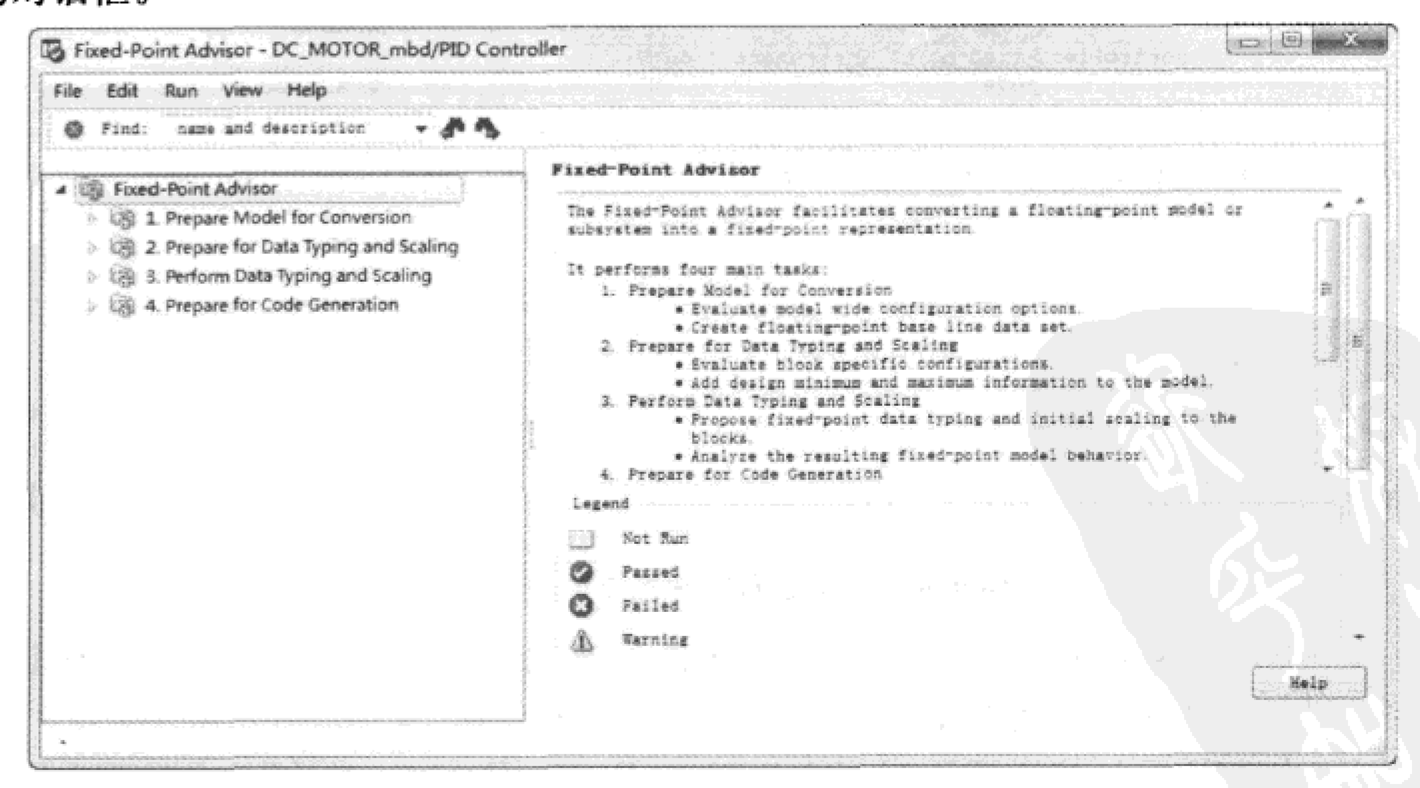


图 9.6.2　Fixed-Point Advisor 界面

其中包括四项任务：①为模型转换作准备(Prepare Model for Conversion)；②为数据类型和自动定标作准备(Prepare for Data Typing and Scaling)；③执行初始数据类型和自动定标(Perform Data Typing and Scaling)；④检查模型的设置是否适合代码的生成(Prepare for Code Generation)。

2. 执行定点化任务

在顶级任务目录 Fixed-Point Advisor 上右击，选择 Run to Failure 选项即可开始定点化工作，如图 9.6.3 所示。

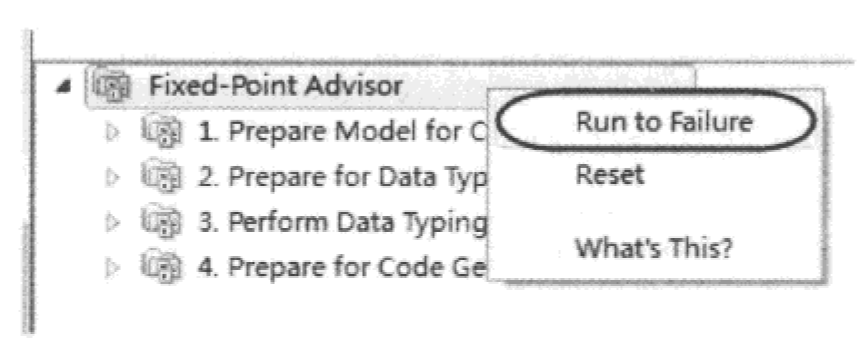


图 9.6.3　执行定点化任务

3. 修改警告及错误

(1) 在用户没有自行对模型做过其他与定标相关的设置的情况下，第一个任务 Verify model Simulation settings 就会失败，如图 9.6.4 所示。报告中详细列出了错误原因和修改意见，用户可单击下方的 Modify All 按钮自动修正错误。然后在对话框的菜单栏选择 Run → Run to Failure 继续执行任务。

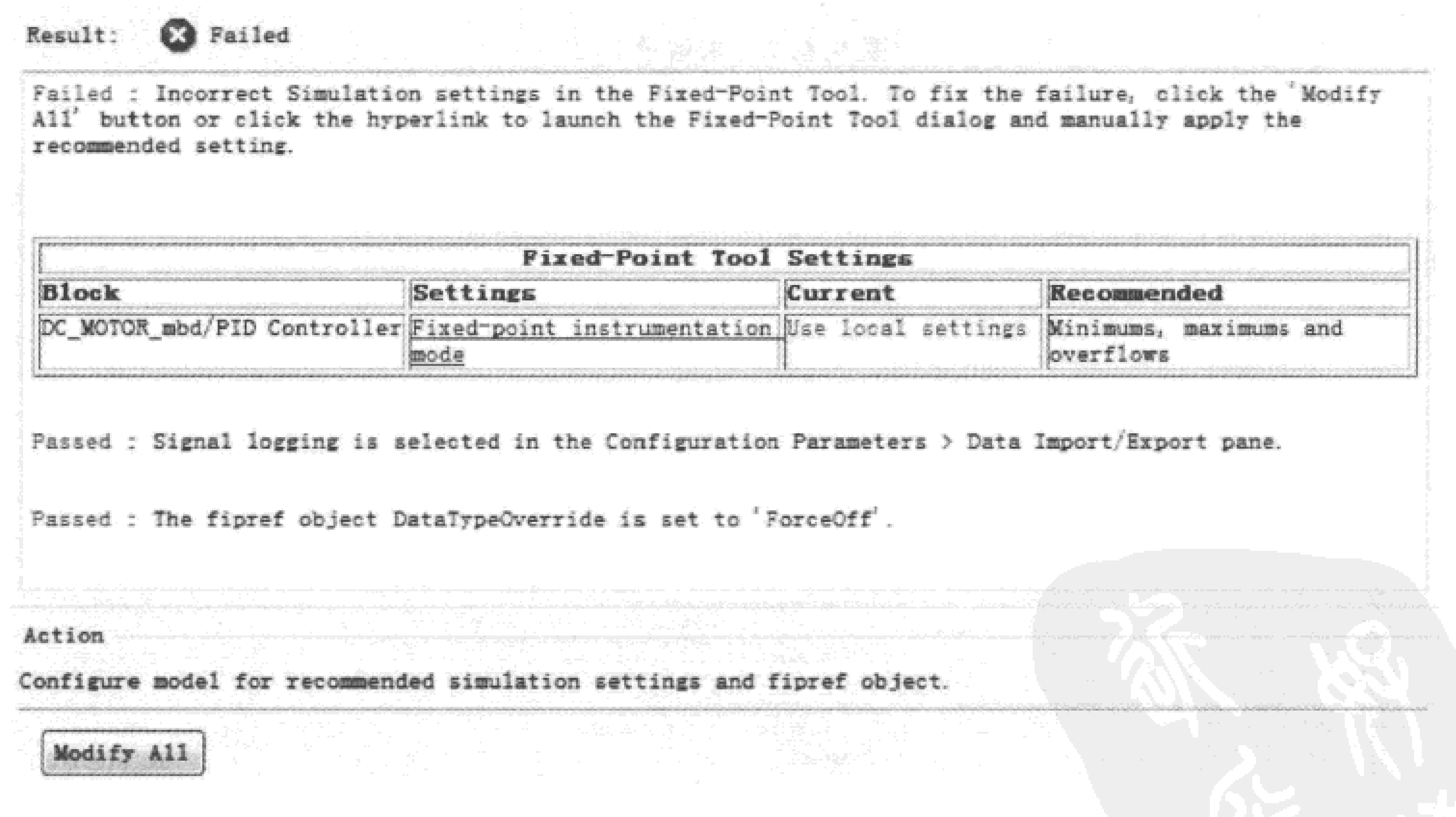


图 9.6.4　验证模型仿真设置任务界面

(2) 任务执行到 set up signal logging 时，再次出现错误，如图 9.6.5 所示。右边页面显示执行任务失败的原因为至少需要指定一个信号的数据被记录。

单击 Explore Result 按钮打开 Model Advisor Result Explore 对话框，在界面中勾选每个模块的 EnableLogging 复选框，如图 9.6.6 所示，然后再次运行该任务。

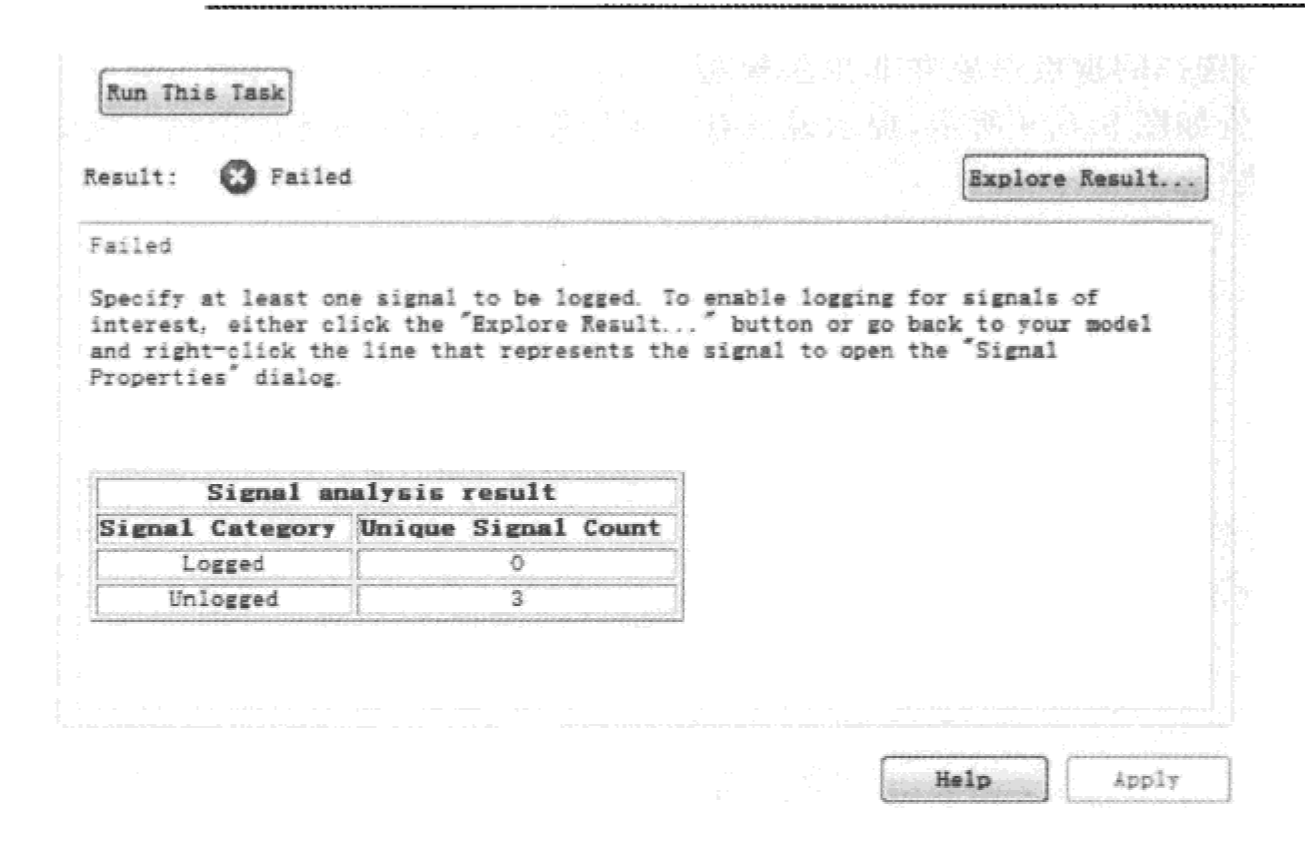


图 9.6.5 设置信号记录任务界面

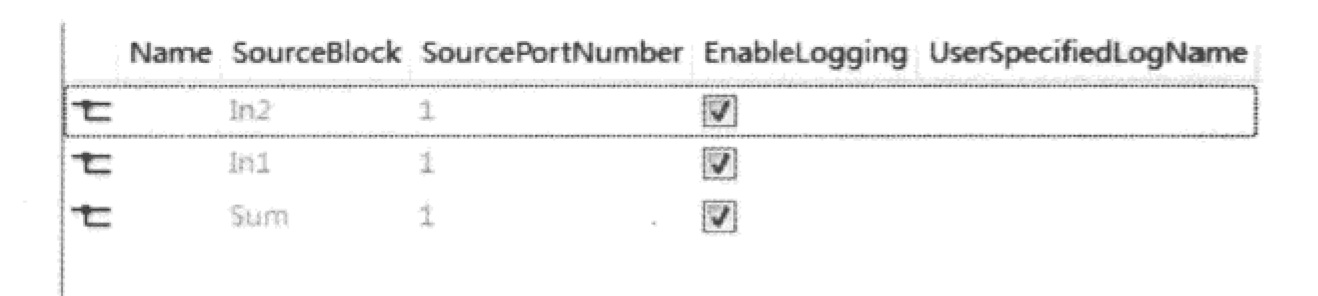


图 9.6.6 设置信号记录

(3) 执行到 1.6 项时,会出现若干警告,例如 1.6.1 项 check model configuration validity diagnostic parameters setting 的警告为 Configuraion Parameters 中的参数设置不合理,如图 9.6.7 所示。

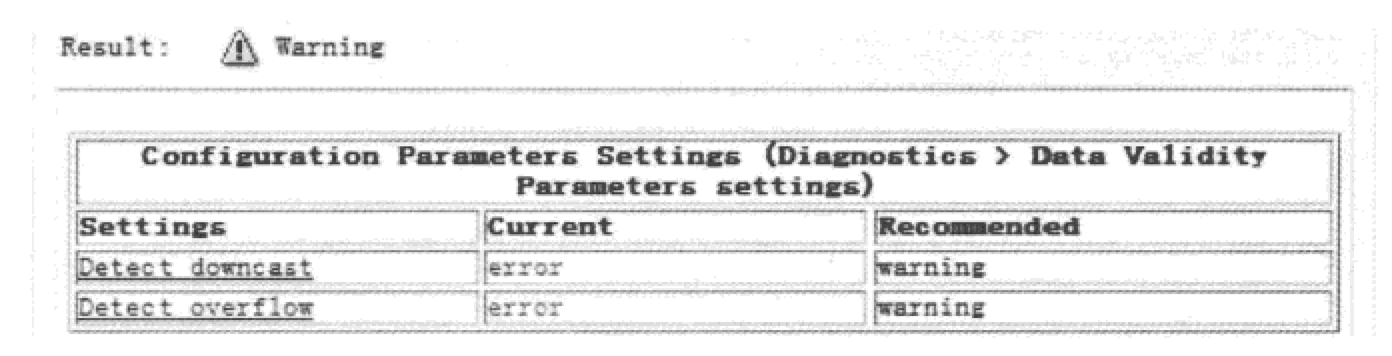


图 9.6.7 检查模型设置与诊断参数设置任务界面

用户可以手动修改这些警告。单击链接 Detect_downcast,打开设置界面,并按照建议设置为 warning,如图 9.6.8 所示。其他警告可按同样方法修改。

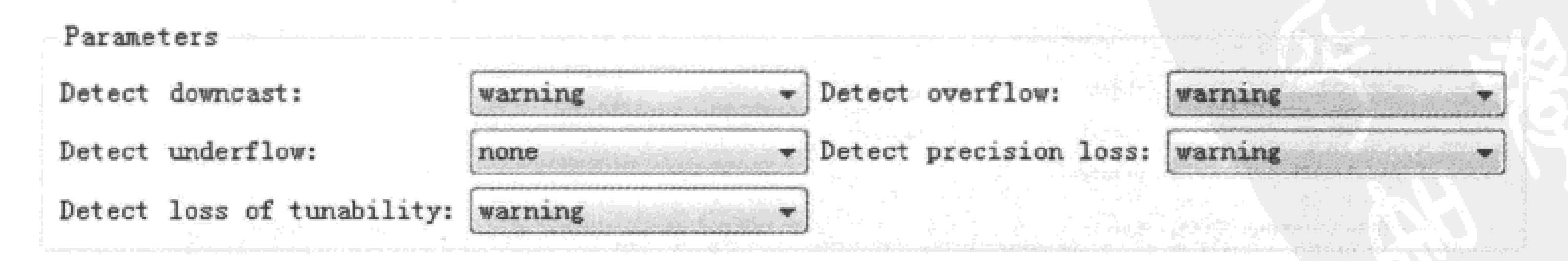


图 9.6.8 修改参数

由于之前选择的顶点对象并非顶层模型，1.6.3 项 check for proper bus usage 的警告无法修改，其警告如图 9.6.9 所示，提示说只有在顶层模型运行该项才能检测。该警告并不影响定标任务的执行。

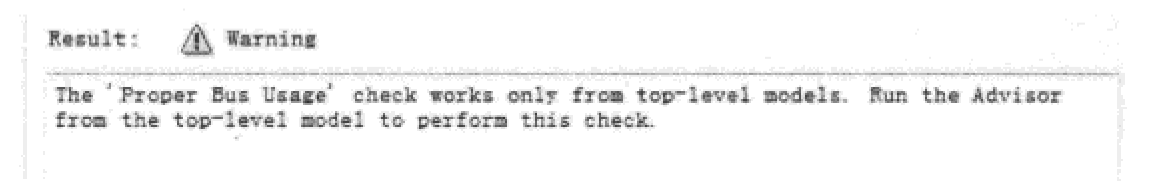
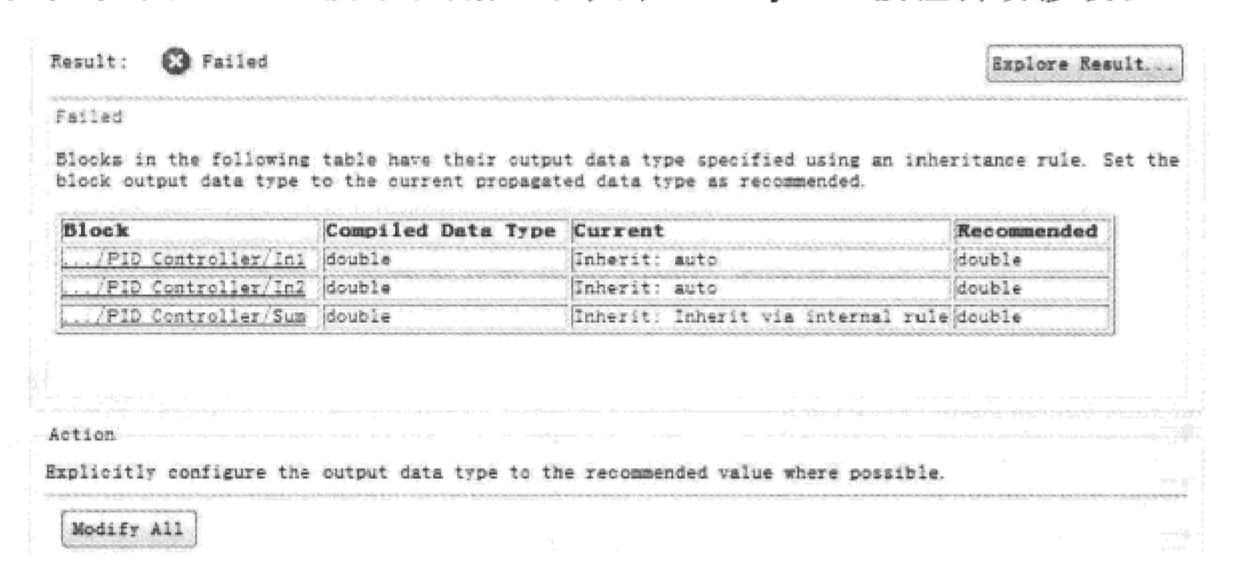


图 9.6.9　检查总线使用情况任务页面

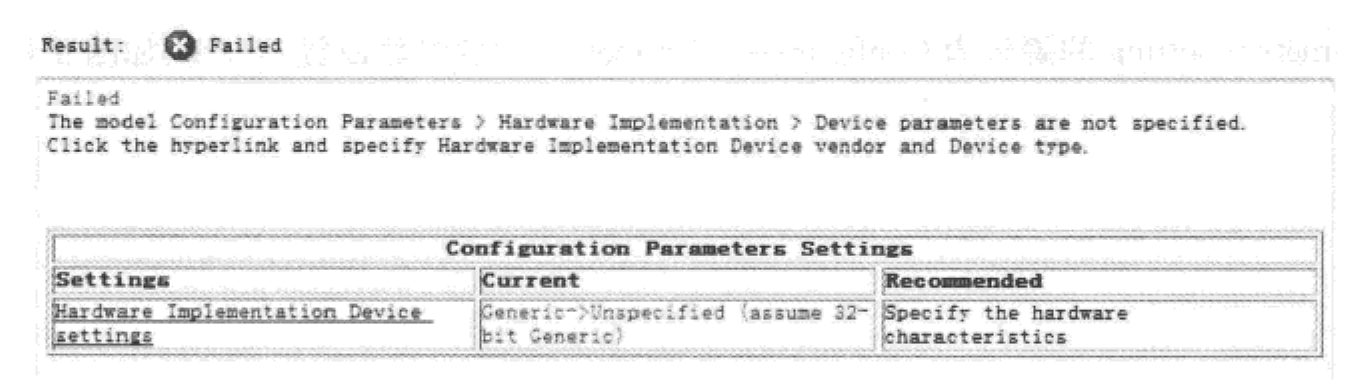
(4) 执行到 2.2 项 Remove output data type inheritance 时出错，提示有若干模块的输出数据类型为继承，如图 9.6.10 所示。用户可单击 Modify All 按钮自动修改。



Block	Compiled Data Type	Current	Recommended
.../PID Controller/In1	double	Inherit: auto	double
.../PID Controller/In2	double	Inherit: auto	double
.../PID Controller/Sum	double	Inherit: Inherit via internal rule	double

图 9.6.10　取消输出数据类型的继承设置任务页面

(5) 执行到 2.5 项 Verify hardware selection 时出错，提示用户选择硬件，如图 9.6.11 所示。考虑到本实验最终要在 ARM 芯片上实现，硬件应选择 ARM7。



Configuration Parameters Settings		
Settings	Current	Recommended
Hardware Implementation Device settings	Generic->Unspecified (assume 32-bit Generic)	Specify the hardware characteristics

图 9.6.11　验证硬件选择任务界面

单击链接 Hardware_Implementation_Device_settings，打开设置界面，选择 ARM7，并设定取整方式(Signed integer division rouds to)为 zero，如图 9.6.12 所示。

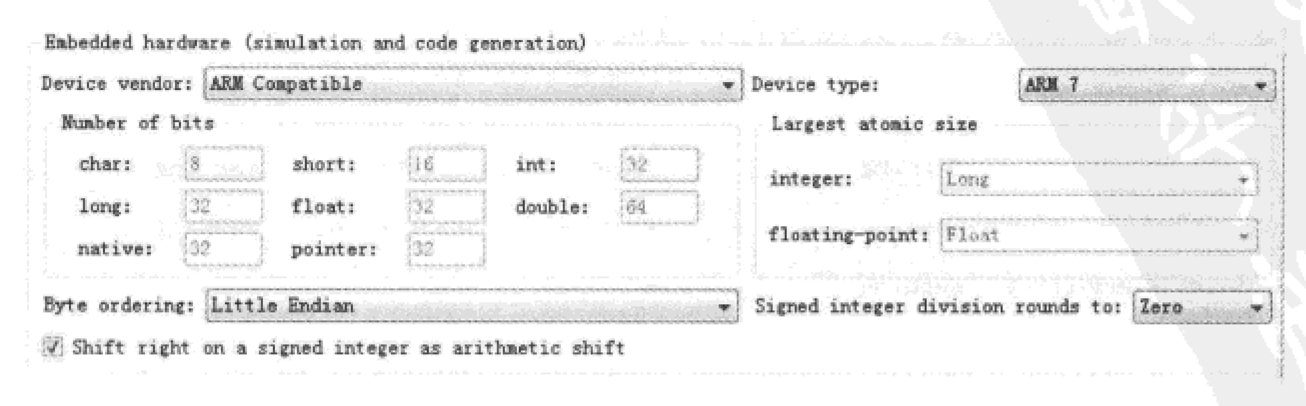
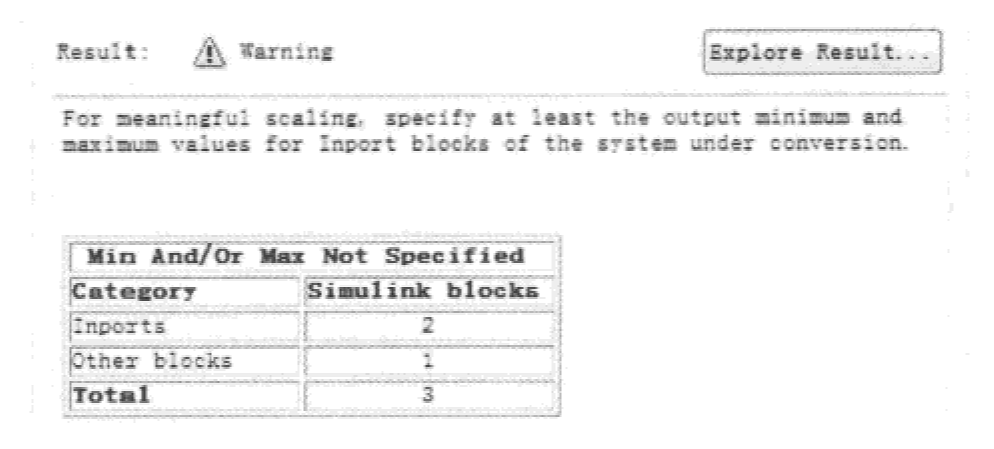


图 9.6.12　Hardware Implementation 界面



(6) 执行到 2.7 项 specify block minimum and maximum values 时出错，提示有若干模块未定义最大值或最小值，如图 9.6.13 所示。



Min And/Or Max Not Specified	
Category	Simulink blocks
Inports	2
Other blocks	1
Total	3

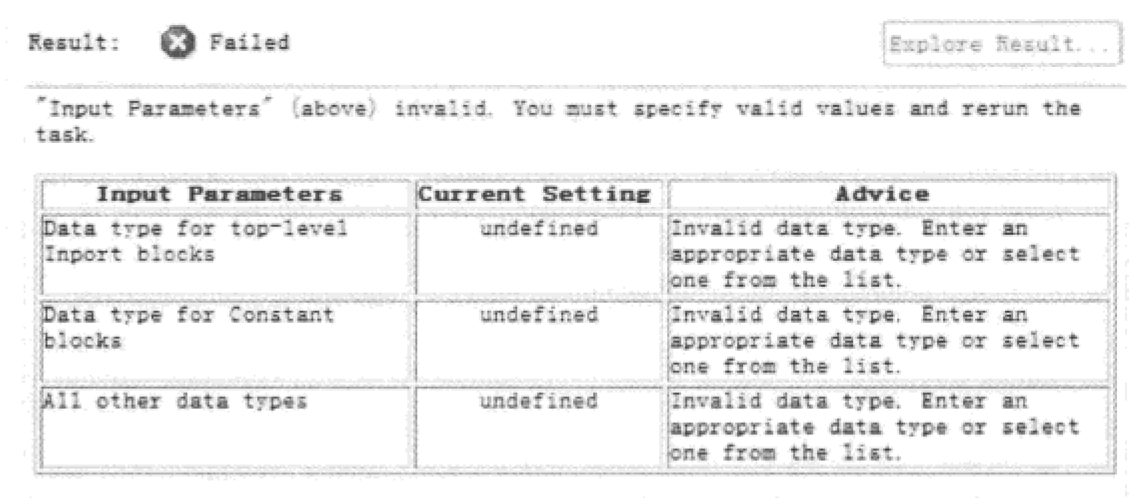
图 9.6.13　指定模块最大/小值任务页面

单击 Explore Result 按钮，打开 Model Advisor Result Explore，如图 9.6.14 所示。根据模块仿真过程中记录的最大值或最小值，容易确定其输出值(OutMin/OutMax)的范围。

Name	BlockType	SimMin	SimMax	OutMin	OutMax
In2	Inport	-15.3539	240.354	[-16]	[245]
In1	Inport	0	225	[0]	[225]

图 9.6.14　指定模块最大/小值

(7) 执行到 3.1 项 Propose data type and scaling 时出错，提示未指定需要的数据类型，如图 9.6.15 所示。



Input Parameters	Current Setting	Advice
Data type for top-level Inport blocks	undefined	Invalid data type. Enter an appropriate data type or select one from the list.
Data type for Constant blocks	undefined	Invalid data type. Enter an appropriate data type or select one from the list.
All other data types	undefined	Invalid data type. Enter an appropriate data type or select one from the list.

图 9.6.15　数据类型及定标任务页面

由于选择的是 ARM7 芯片，为 32 位处理器，因此，将数据类型暂定为有符号 32 位定点数，如图 9.6.16 所示。

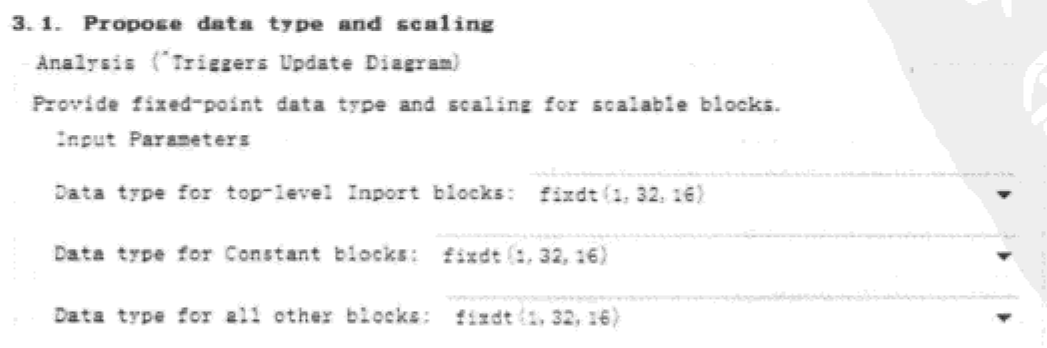


图 9.6.16　指定数据类型

考虑到定点化后，PID Controller 子系统的输入/输出信号数据类型已经与其他模块不符，因此需要在该模块的输入/输出口添加数据类型转换模块，如图 9.6.17 所示。

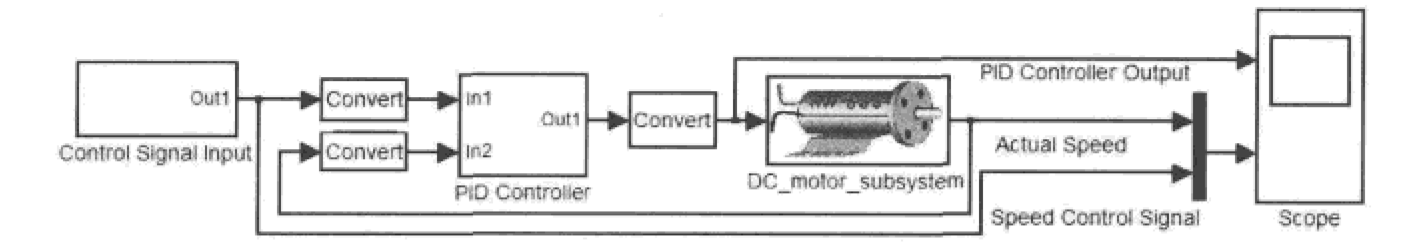


图 9.6.17　添加数据类型转换模块

再次运行这项任务，变为了警告，如图 9.6.18 所示。单击 Modify All 按钮改为系统建议的定标方案。

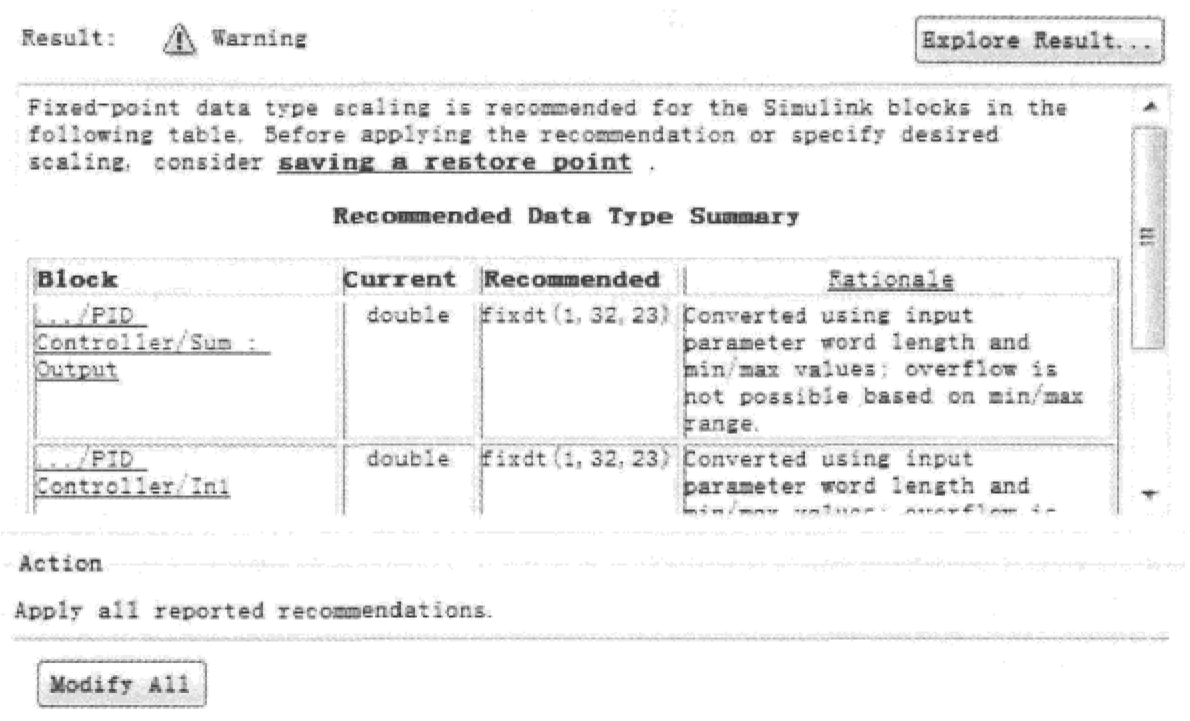


图 9.6.18　数据类型及定标任务页面

(8) 执行到 4.2 项 Identify questionable fixed-point operations 时，出现警告，如图 9.6.19 所示。这项警告是关于定标后数据运算的定标规则的，可以在 Fixed-Point Tool 中对其作进一步修改。

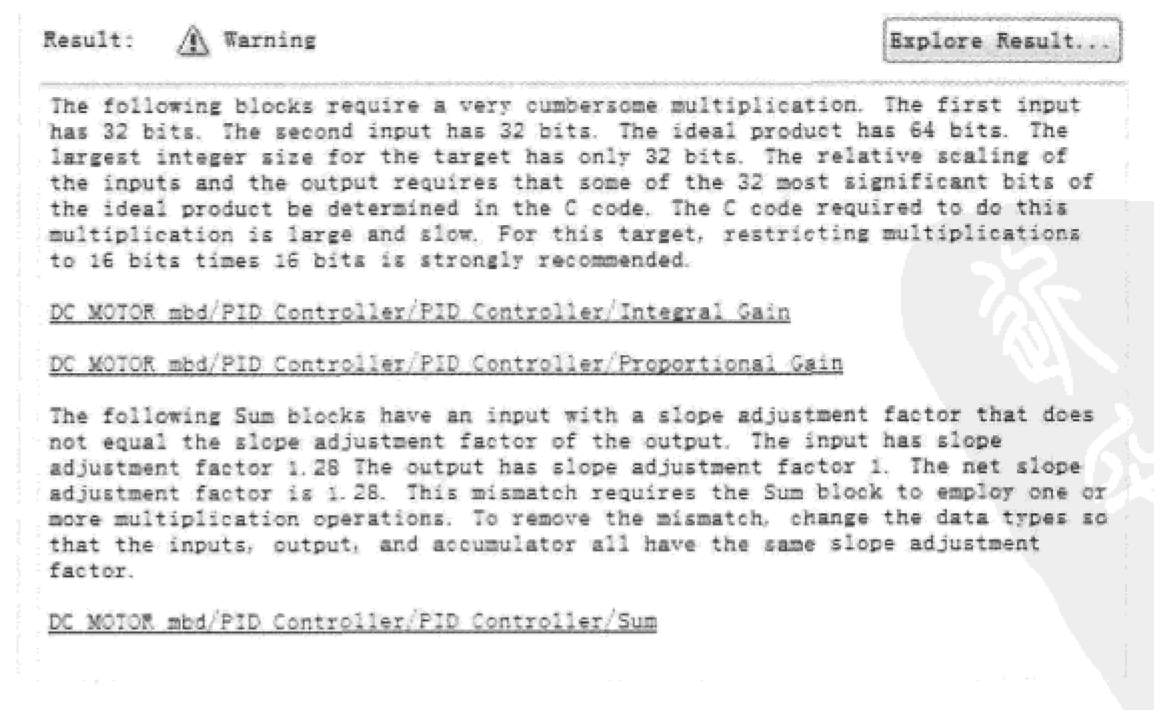
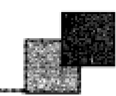


图 9.6.19　存疑的定点化处理任务界面



事实上，若要得到更为精确的定标方案，还需要为 PID Controller 模块的内部参数作定点化处理。打开该模块的 Data Types 界面，将数据类型预设为有符号 32 位定点数，待 Fixed-point Tool 对其优化调整，如图 9.6.20 所示。

Main　PID Advanced　Data Types　State Attributes

	Data Type	Assistant
P parameter:	fixdt(1,32,16)	>>
I parameter:	fixdt(1,32,16)	>>
P product output:	fixdt(1,32,16)	>>
I product output:	fixdt(1,32,16)	>>
Integrator output:	fixdt(1,32,16)	>>
Sum output:	fixdt(1,32,16)	>>
Accumulator of Sum:	fixdt(1,32,16)	>>
Saturation output:	fixdt(1,32,16)	>>

图 9.6.20　PID Controller 模块内部参数定点化

9.6.2　Fixed Point Tools

利用 Fixed-Point Advisor 工具自动定标后，需进一步优化定标，此时需用到 Fixed-Point Tool 工具。在 PID Controller 子系统上右击，选择 Tool→Fixed-Point→Fixed-Point Tool 选项，打开图 9.6.21 所示界面。

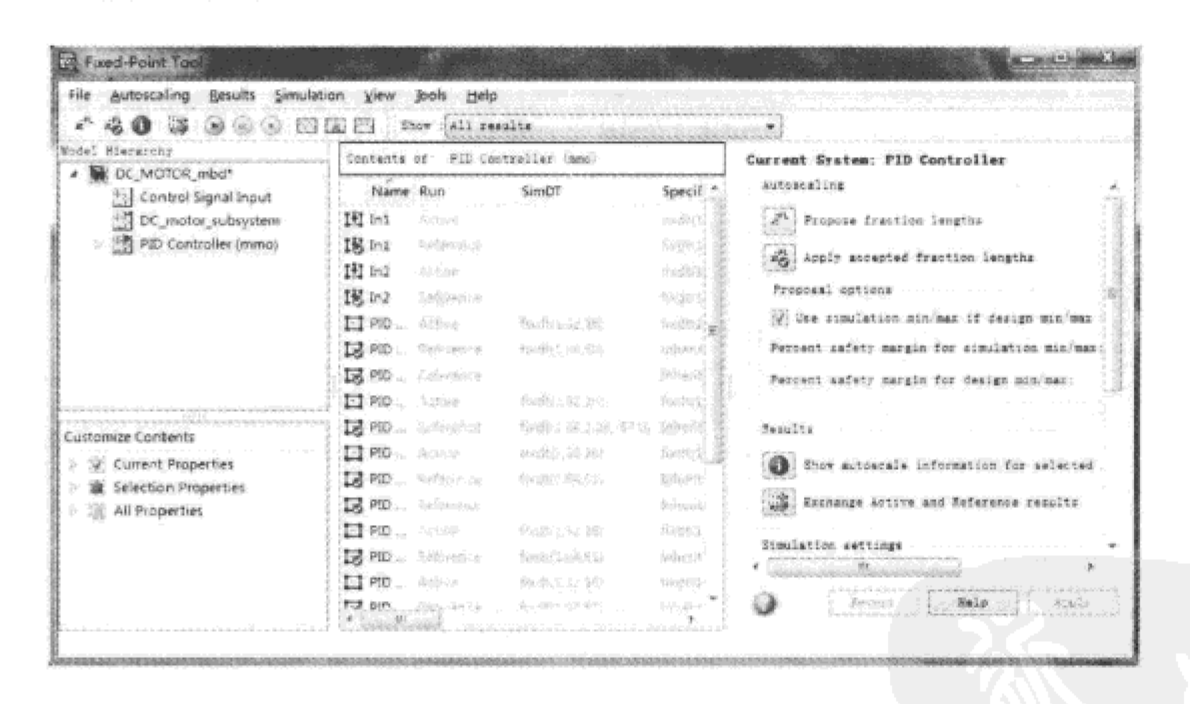


图 9.6.21　Fixed-Point Tool 界面

1. 记录定点/浮点仿真数据

单击 Results 菜单，分别选择 Clear Active Results 和 Clear Reference Results 命令，如图 9.6.22 所示。这两项可完全清除之前运行 Fixed-Point Advisor 时保存的数据，以便 Fixed-Point Tool 生成一组新的数据，进行优化定标工作。

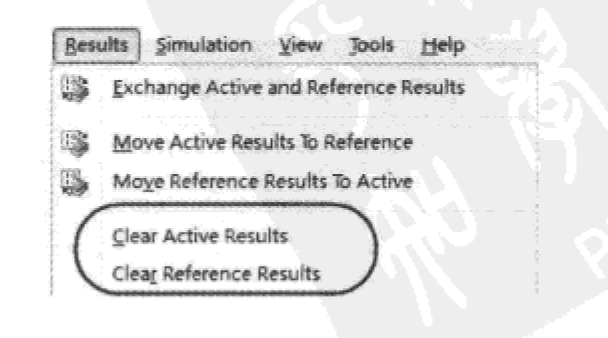


图 9.6.22　清除 Active/Reference 数据

(1) 记录浮点仿真数据。在 Fixed-point instrumentation mode 中选择 Minimums, maximums and overflows 选项,即在运算过程中记录变量的最大值、最小值和溢出次数。在 Data type override 中选择 Double 选项,即在运算过程中运用浮点数,然后按下按钮,如图9.6.23所示。

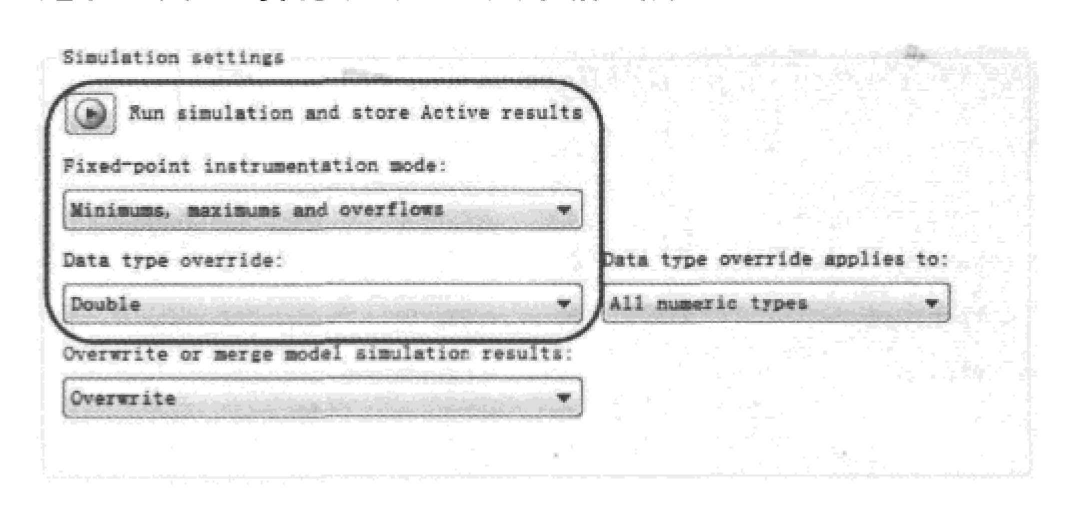


图 9.6.23 记录浮点仿真数据

运行后,得到变量的数据,如图 9.6.24 所示。显然,在使用浮点数时不会发生溢出。



图 9.6.24 浮点数无溢出

选择 Exchange Active and Reference results,将仿真结果保存为参考数据(Reference),如图9.6.25所示。

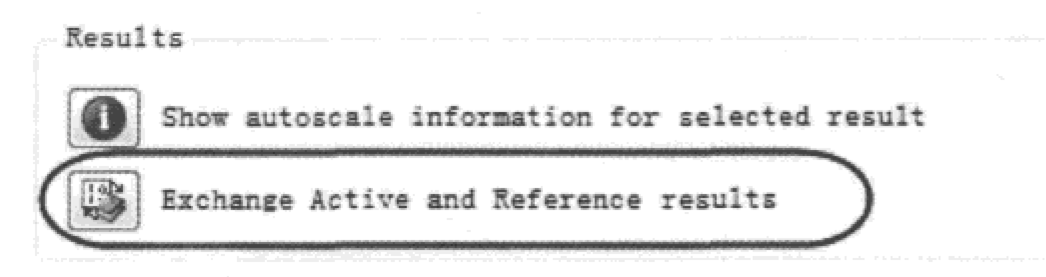


图 9.6.25 将数据转换为 Reference 类型

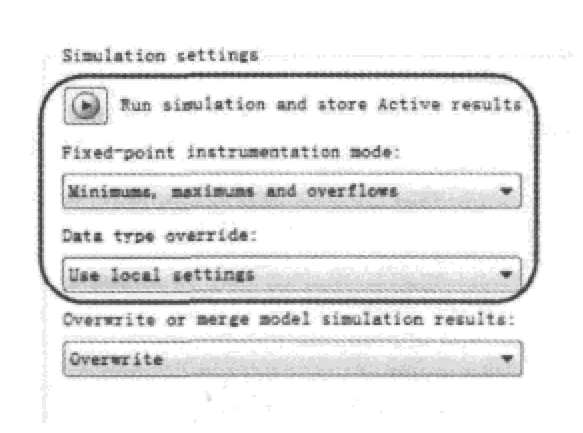
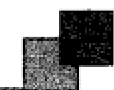
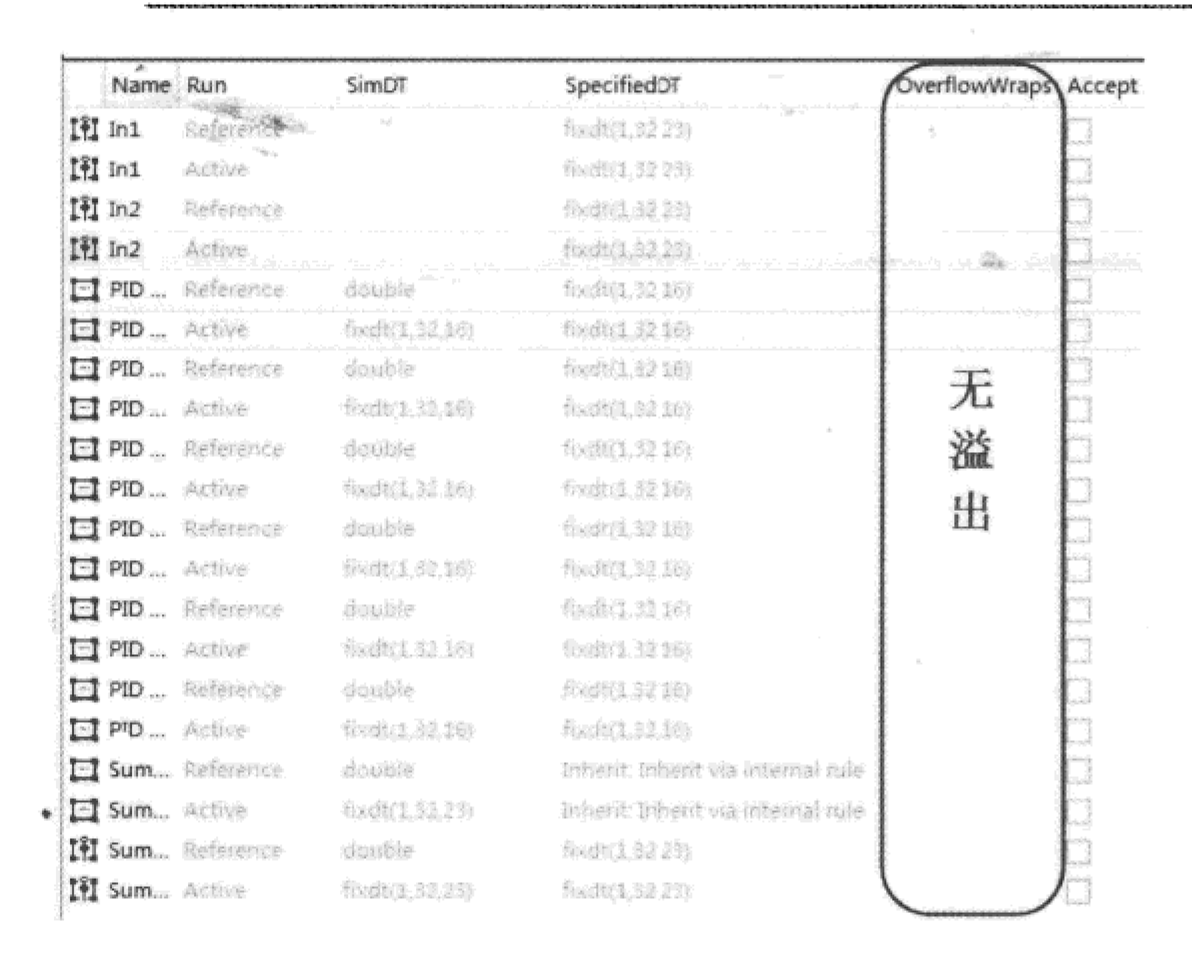


图 9.6.26 记录定点仿真数据

(2) 记录定点仿真数据。在 Data type override 中选择 Use Local Setting 选项,即在运算过程中运用现有定标方案,然后单击按钮,如图 9.6.26 所示。

运行后,得到变量的数据,如图 9.6.27 所示。每个变量都有 Active 和 Reference 两组数据,标为 Active 属性的是定点数据。可看到 OverflowWraps 栏没有值,即在运算过程中没有数据溢出,这说明自动定标的结果正确。





Name	Run	SimDT	SpecifiedDT	OverflowWraps	Accept
In1	Reference		fixdt(1,32,23)		
In1	Active		fixdt(1,32,23)		
In2	Reference		fixdt(1,32,23)		
In2	Active		fixdt(1,32,23)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
PID ...	Reference	double	fixdt(1,32,16)		
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,16)		
Sum...	Reference	double	Inherit: Inherit via internal rule		
Sum...	Active	fixdt(1,32,23)	Inherit: Inherit via internal rule		
Sum...	Reference	double	fixdt(1,32,23)		
Sum...	Active	fixdt(1,32,23)	fixdt(1,32,23)		

图 9.6.27　定点数无溢出

2. 优化定标

分别得到定点/浮点两组数据后，可以方便地观察定点化所带来的误差。以 Sum 模块为例，右击 Sum，在菜单中选择 Time Series Difference(A-R) Plot，命令如图 9.6.28 所示，绘制出两组 Sum 信号的误差曲线。

Sum 信号的误差曲线如图 9.6.29 所示，在 PID Controller/Sum 中，Active 和 Reference 两条曲线基本重合；Difference between Active and Reference Runs 中的曲线则描述了在两条曲线在每个时间点处的误差值，可见最大误差为 1.2×10^{-4}左右。

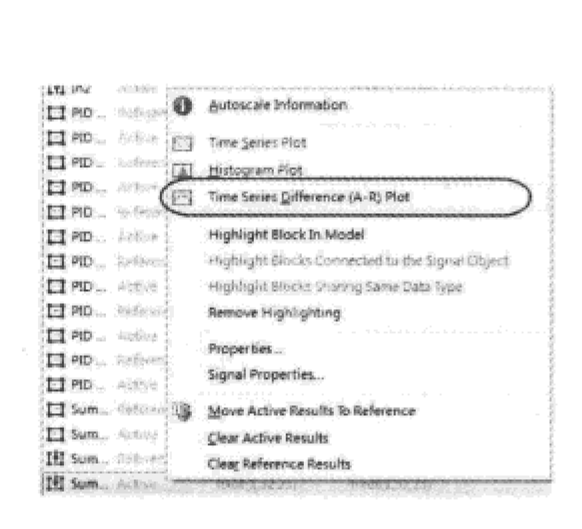


图 9.6.28　绘制误差曲线

图 9.6.29　误差曲线

如果用户对 Fixed-Point Advisor 自动定标的误差情况并不满意，可以单击 Propose fraction lengths 按钮，让 Fixed-Point Tool 设计出一个更加优化的定标方案，若用户认为该方案可接受，则单击 Apply accepted fraction

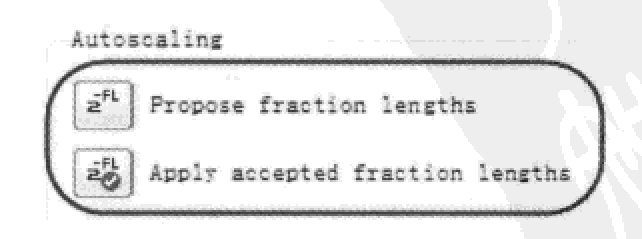


图 9.6.30　系统推荐定标按钮

lengths 按钮，应用此定标方案，如图 9.6.30 所示。

修改完毕后即可发现，Active 属性的模块的定标方案已变化，如图 9.6.31 所示。

Name	Run	SimDT	SpecifiedDT	OverflowWraps
In1	Reference		fixdt(1,32,23)	
In1	Active		fixdt(1,32,23)	
In2	Reference		fixdt(1,32,23)	
In2	Active		fixdt(1,32,23)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,20)	
PID ...	Active		fixdt(1,32,27)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,21)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,20)	
PID ...	Active		fixdt(1,32,28)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,20)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,20)	
PID ...	Reference	double	fixdt(1,32,16)	
PID ...	Active	fixdt(1,32,16)	fixdt(1,32,20)	
Sum...	Reference	double	Inherit: Inherit via internal rule	
Sum...	Active	fixdt(1,32,23)	Inherit: Inherit via internal rule	
Sum...	Reference	double	fixdt(1,32,23)	
Sum...	Active	fixdt(1,32,23)	fixdt(1,32,23)	

图 9.6.31　应用系统推荐定标

此时重复“1. 记录定点/浮点仿真数据”的工作，并再次绘制 Sum 模块的误差曲线如图 9.6.32所示。可见两条曲线几乎完全重合，误差的峰值为 3.5×10^{-6} 左右，精度比优化前提高了两个数量级。

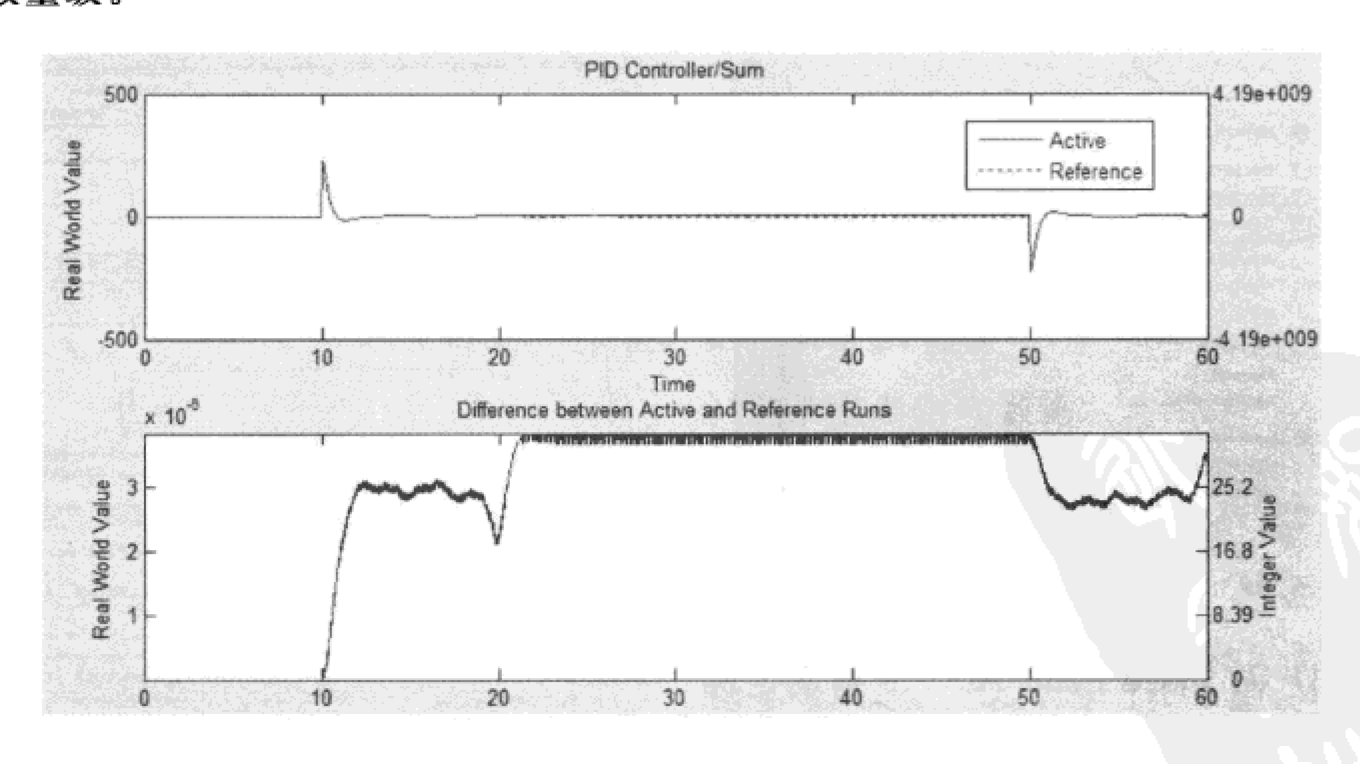
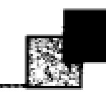


图 9.6.32　误差曲线

运行定点化模型，其仿真结果如图 9.6.33 所示。与未作定点化处理的模型仿真曲线相比，几乎完全一致，这再次证明了 Fixed-Point Advisor/Tool 的定标是合理的，误差控制性能良好。



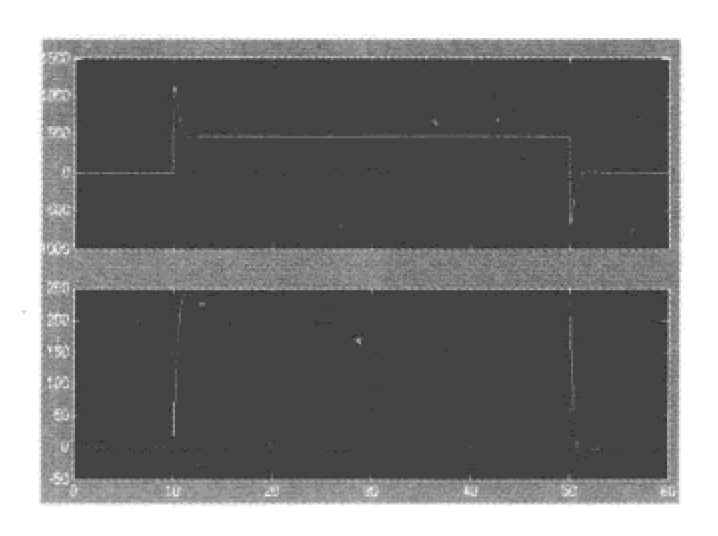


图 9.6.33　仿真曲线

9.7　软件在环测试

软件在环测试是对模型生成的嵌入式C代码进行测试。即在模型环境中对被控对象模型生成的代码进行非实时性检查，以评估生成代码的优劣，完成对模型设计的早期验证。

下面对 PID Controller 子系统作软件在环测试。

(1) 在一个新模型中将 PID Controller 子系统独立出来，用输入/输出模块连接到子系统如图 9.7.1 所示。

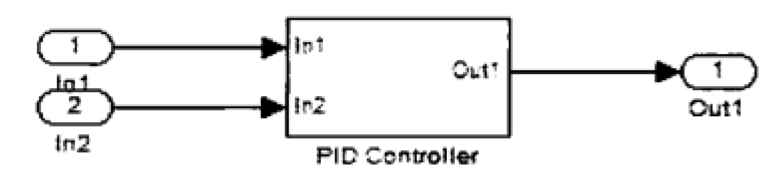


图 9.7.1　PID Controller 子系统

由于 PID Controller 子系统已做过定点化处理，输入/输出端口的数据类型也要作相应调整。根据定标方案，把 In1、In2 模块的数据类型设为 fixtd(1,32,23)，把 Out1 模块的数据类型设为 fixtd(1,32,20)。

(2) 选择 Simulation→Configutation Parameters，打开模型的参数设置对话框，在 solver 列表框选择定步长离散求解器，如图 9.7.2 所示。

Solver options

Type: Fixed-step　　Solver: discrete (no continuous states)

Fixed-step size (fundamental sample time):　auto

图 9.7.2　求解器设置

在 Real-Time Workshop 界面选择 ert. tlc 选项，如图 9.7.3 所示。

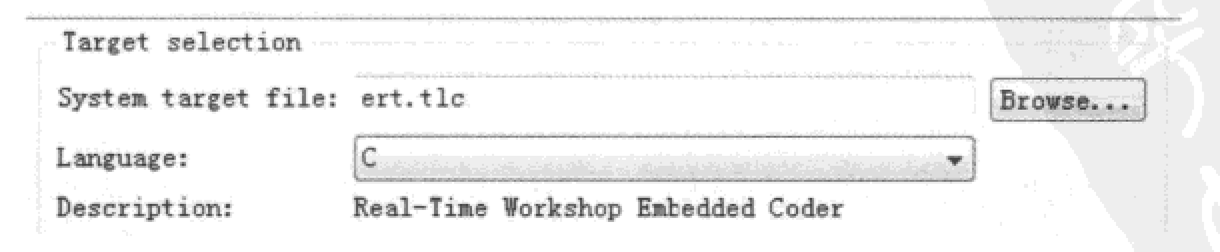


图 9.7.3　设置 tlc

在 Real-Time Workshop→SIL or PIL Verification 页面选择 SIL 选项，如图 9.7.4 所示。

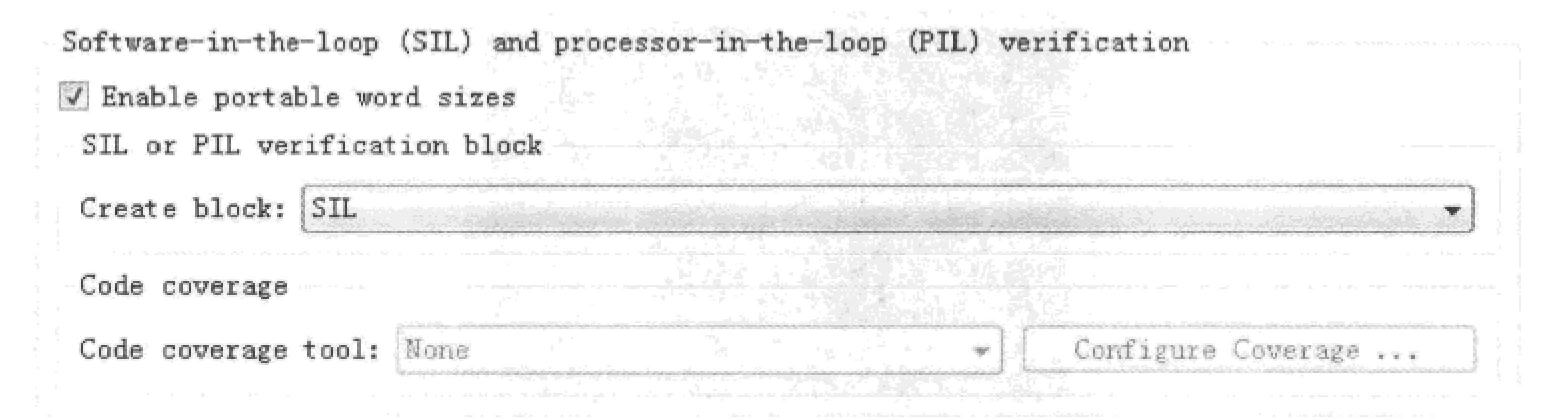


图 9.7.4　设置 SIL

(3) 保存设置后单击按钮生成 SIL 模块。用生成的 SIL 模块替换原模型中的 PID Controller 模块，实现软件在环测试，如图 9.7.5 所示。

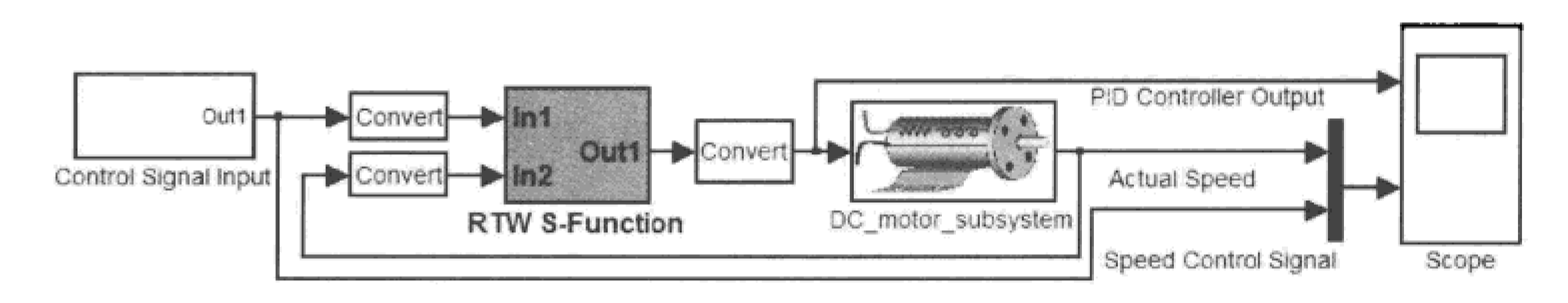


图 9.7.5　SIL 验证模型

仿真结果如图 9.7.6 所示，仿真波形与 PID Controller 控制下的响应相同，说明它们实现的功能一致，即生成的代码能实现模型的功能。

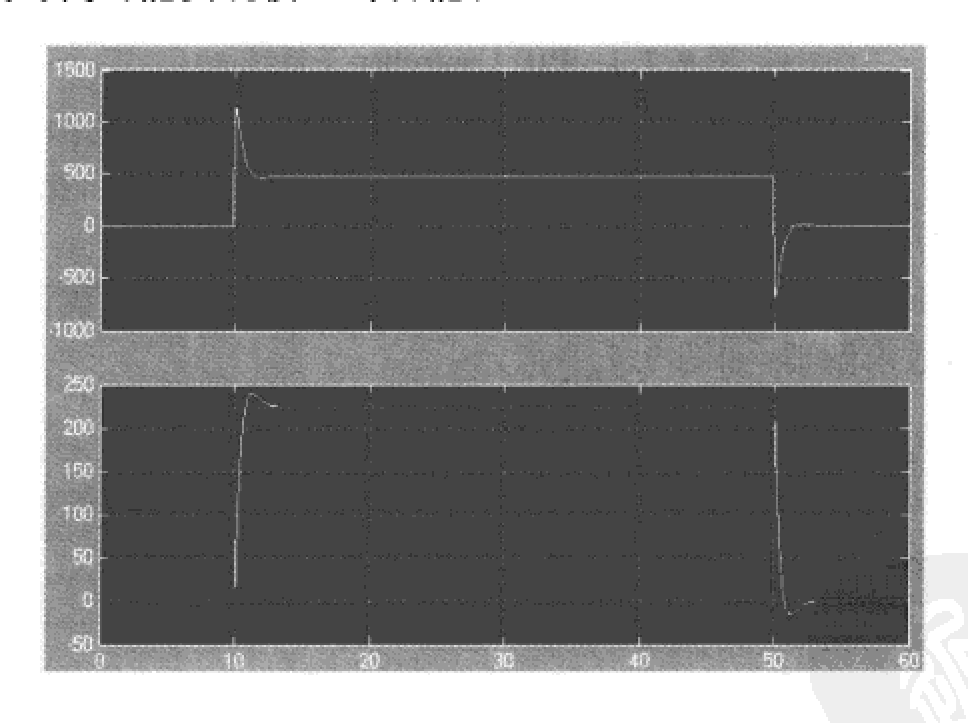
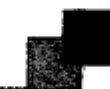


图 9.7.6　仿真波形

9.8　代码跟踪

需求与模型间可建立双向跟踪，代码与模型同样也可以建立这样的跟踪，用户可以通过代码与模型间的链接，快速定位某个模块所对应的代码段，也可以通过分析代码，改进模型。

右击 PID Controller 子系统，选择 subsystem parameters 选项，在弹出的对话框中选择 Treat as



atomic unit 选项,将子系统原子化,如图 9.8.1 所示。关于原子化的问题在 9.9 节介绍。

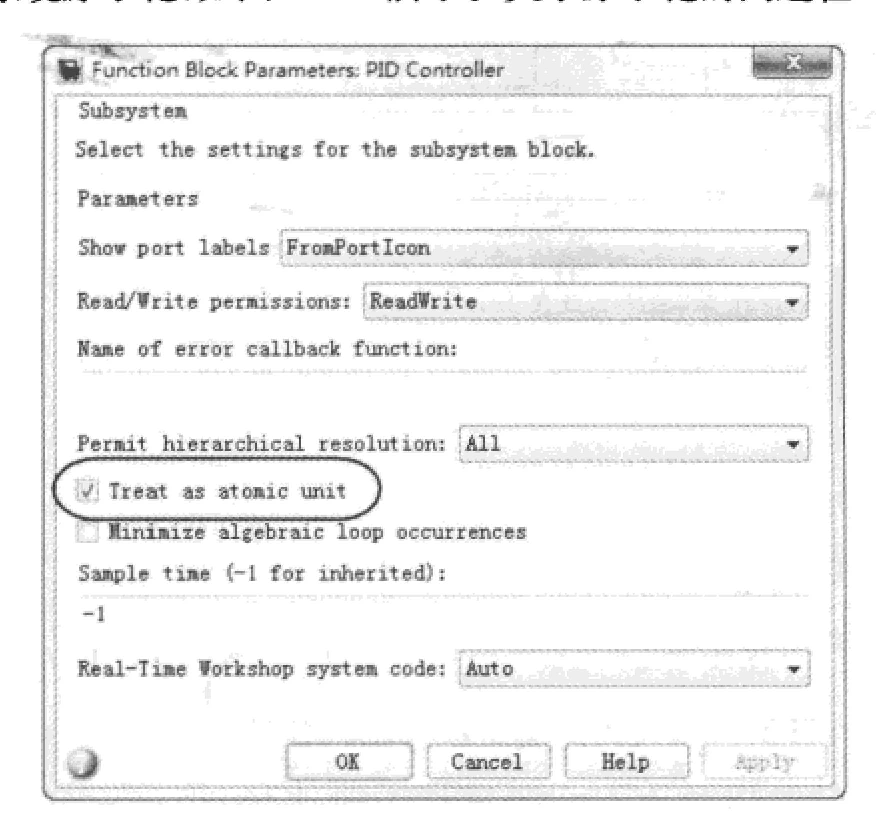


图 9.8.1 模型原子化

在参数设置对话框的 Report 页面,按图 9.8.2 所示进行设置,则在编译模型时自动打开代码生成的报告,并建立模型与代码间的双向关联。

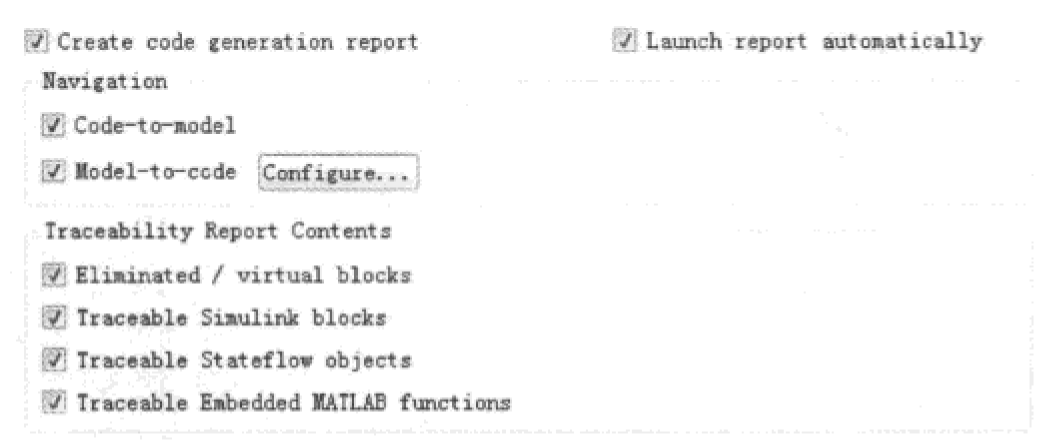


图 9.8.2 报告设置页面

单击按钮编译后自动弹出代码生成报告,在模型中右击 PID Controller 子系统,选择 Real-Time Workshop→Navigate to code 即可连接到对应的代码段,如图 9.8.3 所示。

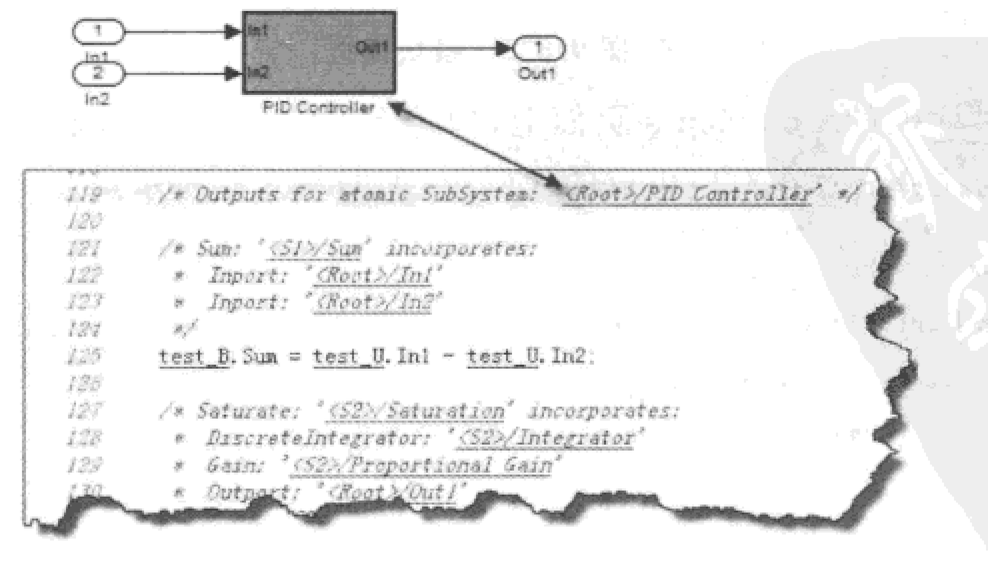


图 9.8.3 模块与代码间的双相跟踪(PID Controller)

单击代码中的链接同样可以定位到响应的模块，如图 9.8.4 所示。

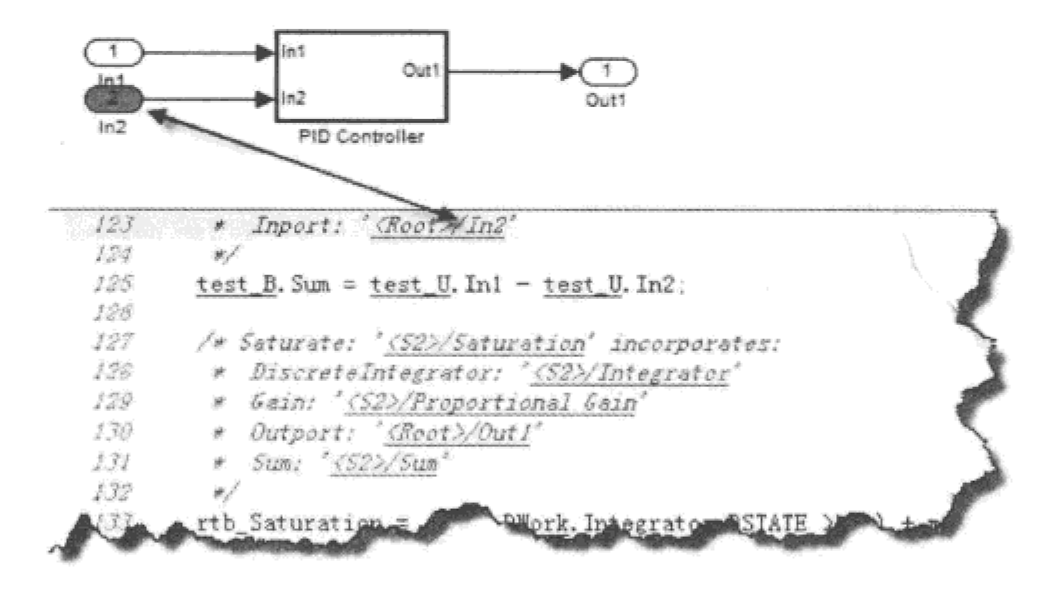


图 9.8.4 模块与代码间的双相跟踪(In2)

为了增加生成代码的可读性，还可以对某一模块进行描述。例如对 PID Controller 子系统进行描述，右击该模块，选择菜单项 Block Properties，在对话框的 Description 栏输入：This block can produce the PID control signal from the error of actual and demand speed，如图9.8.5所示。

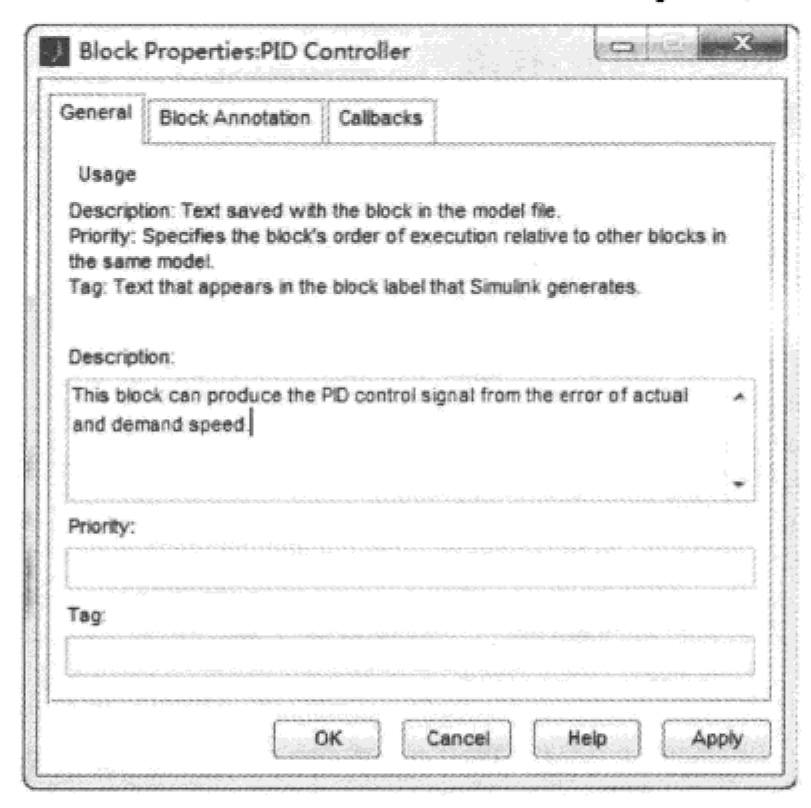


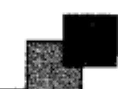
图 9.8.5 模块描述

在参数设置对话框的 Comments 页面上，勾选 Simulink block descriptions 复选框，则对模块的描述会作为注释显示在对应生成的代码中，如图 9.8.6 所示。

Custom comments

- [x] Simulink block descriptions
- [ ] Simulink data object descriptions
- [ ] Custom comments (MPT objects only)
- [ ] Stateflow object descriptions
- [ ] Requirements in block comments
- [ ] MATLAB function help text

图 9.8.6 Comments 页面



单击按钮 生成代码,自动打开代码报告。在模型中右击 PID Controller 模块,在弹出的菜单中选择 Real-Time Workshop → Navigate to Code 命令,就跟踪到该模块对应的代码,如图 9.8.7所示,可看到对模块的描述显示在代码中。

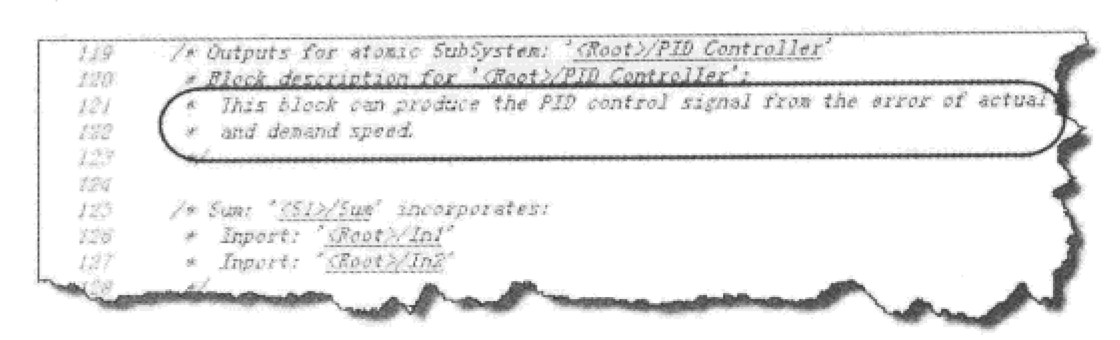


图 9.8.7 模块注释

9.9 代码优化及代码生成

9.9.1 子系统原子化

子系统模块可分为虚拟子系统或非虚拟子系统,主要的不同在于非虚拟子系统在执行仿真时,是被看作一个单元,节省了大量用于中间变量的存储器。而虚拟子系统,只是在视觉上简化了模型。将各个功能模块合并成一个非虚拟子系统,可提高代码执行效率。

用户也可以在添加子系统模块时,直接添加 Ports & Subsystems 子库里的 Atomic Subsystem 模块,然后把需要创建子系统的模块拖入 Atomic Subsystem 模块,如图 9.9.1 所示。它与 Subsystem 模块的差别在于,前者默认启用了 Treat as atomic unit 选项。从外观上看,启用了 Treat as atomic unit 选项的子系统模块,边框加粗了。

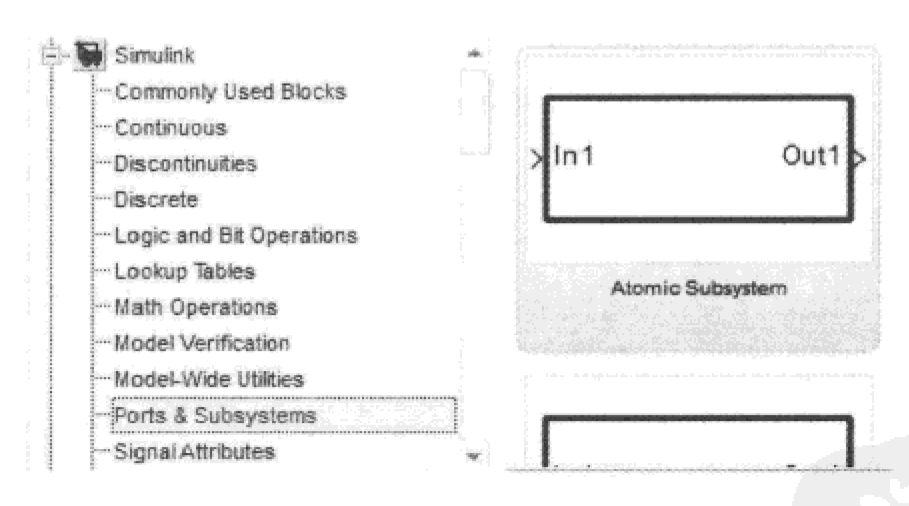


图 9.9.1 子系统模块

选中模型菜单项 Format → Block Displays → Sorted Order,在仿真时,用户可以从模块右上角的数字看出,虚拟子系统模块与非虚拟子系统模块在执行过程中的差别。设置为原子化子系统之前,模型标号如图 9.9.2 所示。

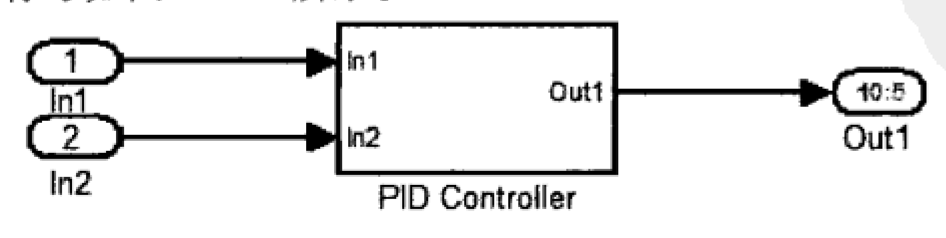


图 9.9.2 虚拟子系统

设为原子化子系统之后，模型标号如图 9.9.3 所示。

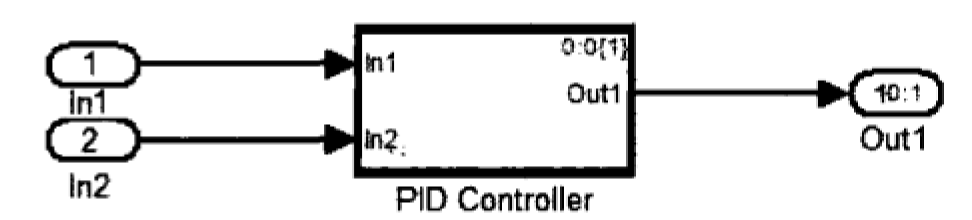


图 9.9.3 原子化子系统

9.9.2 确定芯片类型

执行 Model Advisor 检查时，根据 Check the hardware implementation 检查项的提示，如果用户需要生成针对具体芯片的代码，则需要在模型参数配置窗口指定硬件类型、TLC 文件、芯片库文件，如图 9.9.4 所示。

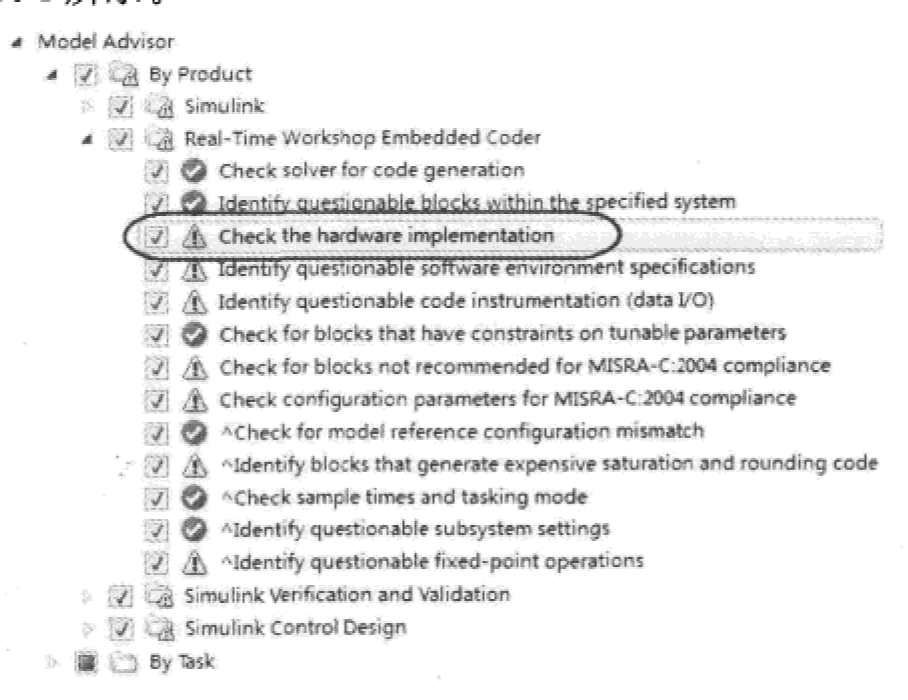


图 9.9.4 Model Advisor 任务树

在模型参数配置窗口的 hardware implementation 部分，指定硬件类型，如图 9.9.5 所示。

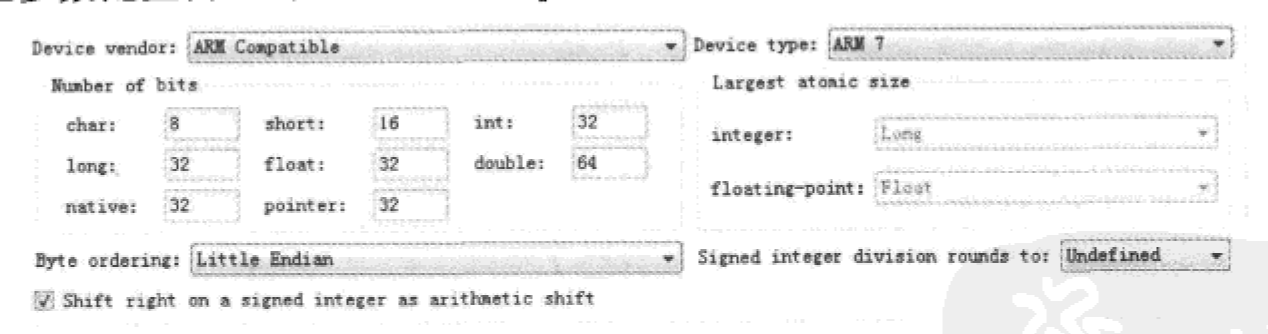


图 9.9.5 指定硬件类型

在模型参数配置窗口的 Real-Time Workshop 部分，设置 TLC 文件为 ert.tlc，如图 9.9.6 所示。

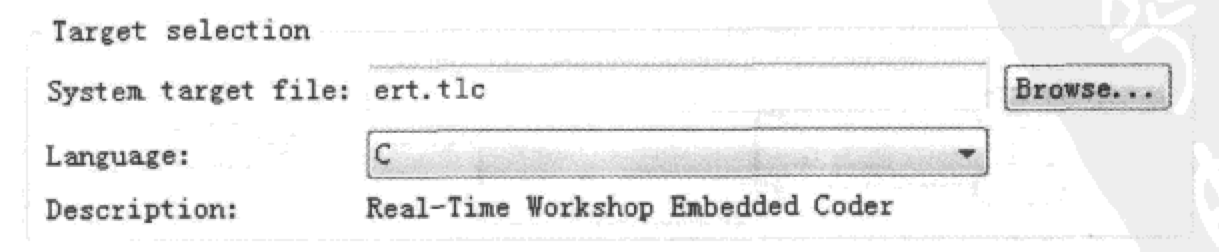
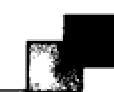


图 9.9.6 指定 TLC 文件



9.9.3　代码检查

在生成代码之前，用户还可以根据项目的要求，选择代码生成的目标，然后执行 Code Generation Advisor 检查，根据建议进行修改，以改进代码，尽量符合用户选定的目标。

1. 选择代码生成目标

上一节已经设置了基于 ert 的 TLC 文件，之后在 Real-Time Workshop 设置页面的下半部，单击 Set objectives... 按钮，选择代码生成目标，如图 9.9.7 所示。

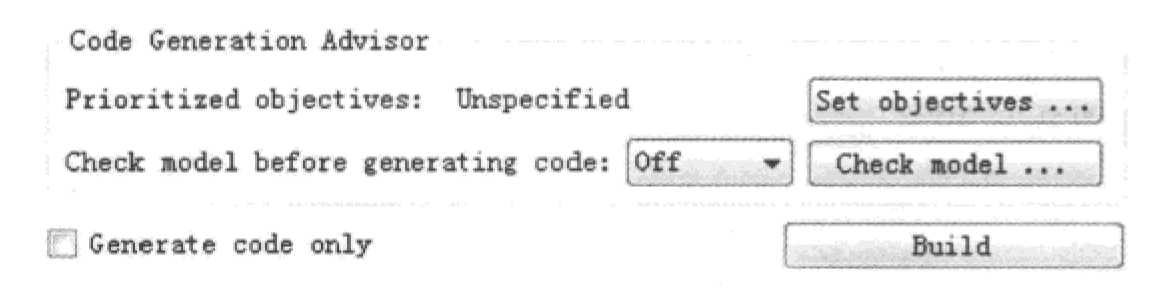


图 9.9.7　模型参数配置窗口

单击左箭头按钮，将左侧窗口的目标添加到右侧窗口，或单击右箭头按钮删除已添加的目标。

根据重要性单击上下箭头按钮，在右侧窗口，将选中的各目标排序，如图 9.9.8 所示。代码目标及意义如表 9.9.1 所列。

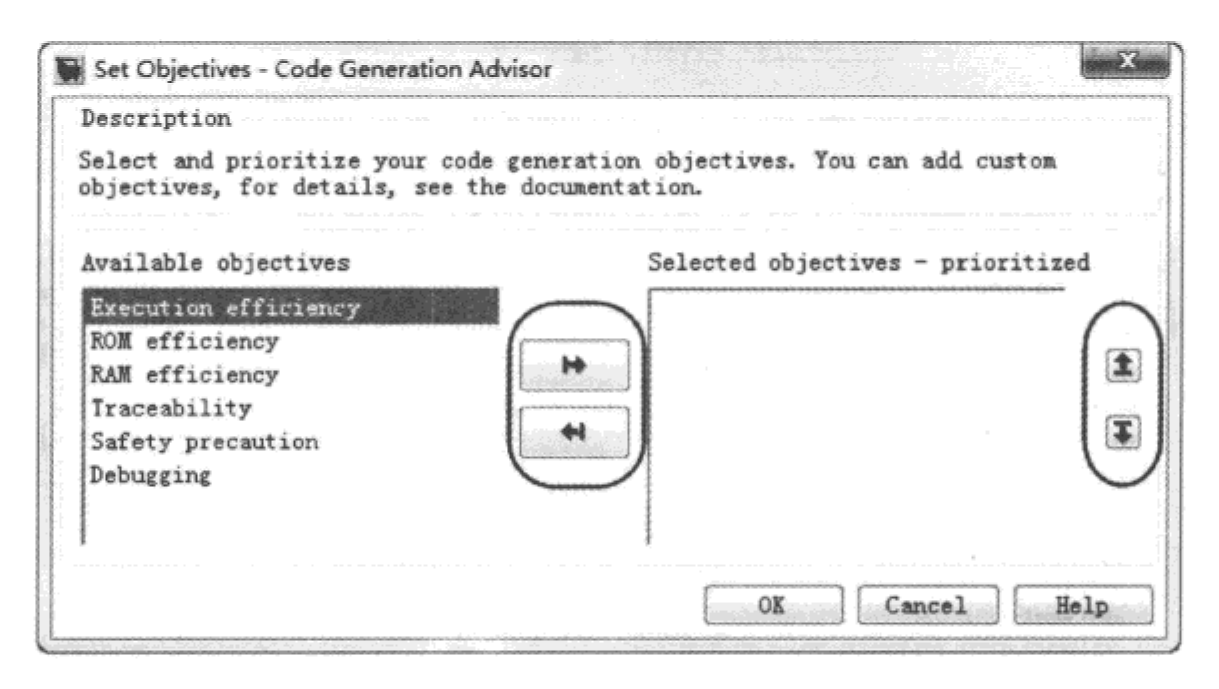


图 9.9.8　选择代码生成目标

表 9.9.1　代码目标及意义

目　标	意　义	目　标	意　义
Execution efficiency	提高执行效率	ROM efficiency	减少 ROM 使用量以及执行时间
RAM efficiency	减少 RAM 使用量以及执行时间	Traceability	提供模型元素与代码之间的映射关系
Safety precaution	加强代码的清晰度、确定度、健壮性、可验证性	Debugging	调试代码生成过程

2. Code Generation Advisor 检查

(1) 本例选中代码目标为 Execution efficiency、ROM efficiency、RAM efficiency 与 Traceability；

(2) 单击 Check model... 按钮，如图 9.9.9 所示。

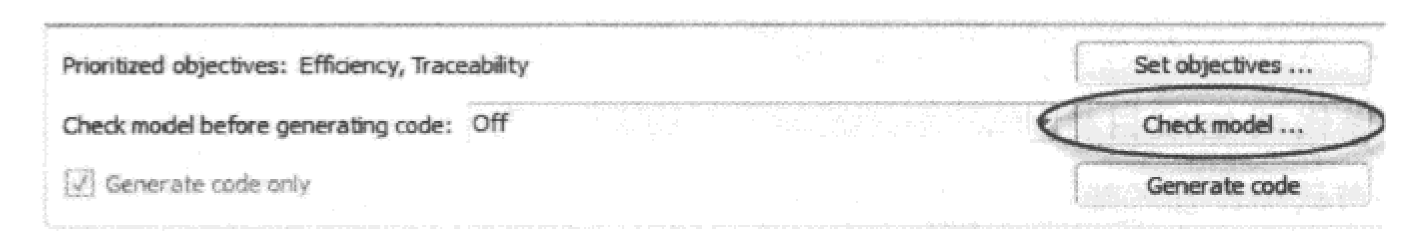


图 9.9.9　Code Generation Advisor 检查

(3) 在系统选择窗口，选中整个模型，而不是某个子系统，确定后开始 Code Generation Advisor 检查，如图 9.9.10 所示。

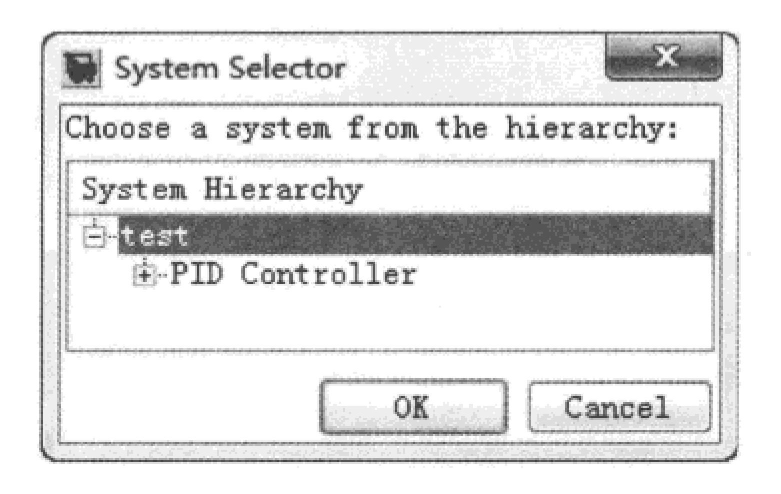


图 9.9.10　System Selector 界面

(4) 运行 Advisor 后，出现部分警告项，例如，check model configuration settings against code generation objectives 项，列出部分未优化的参数，可通过底部的 Modify Parameters 按钮自动修改，如图 9.9.11 所示。

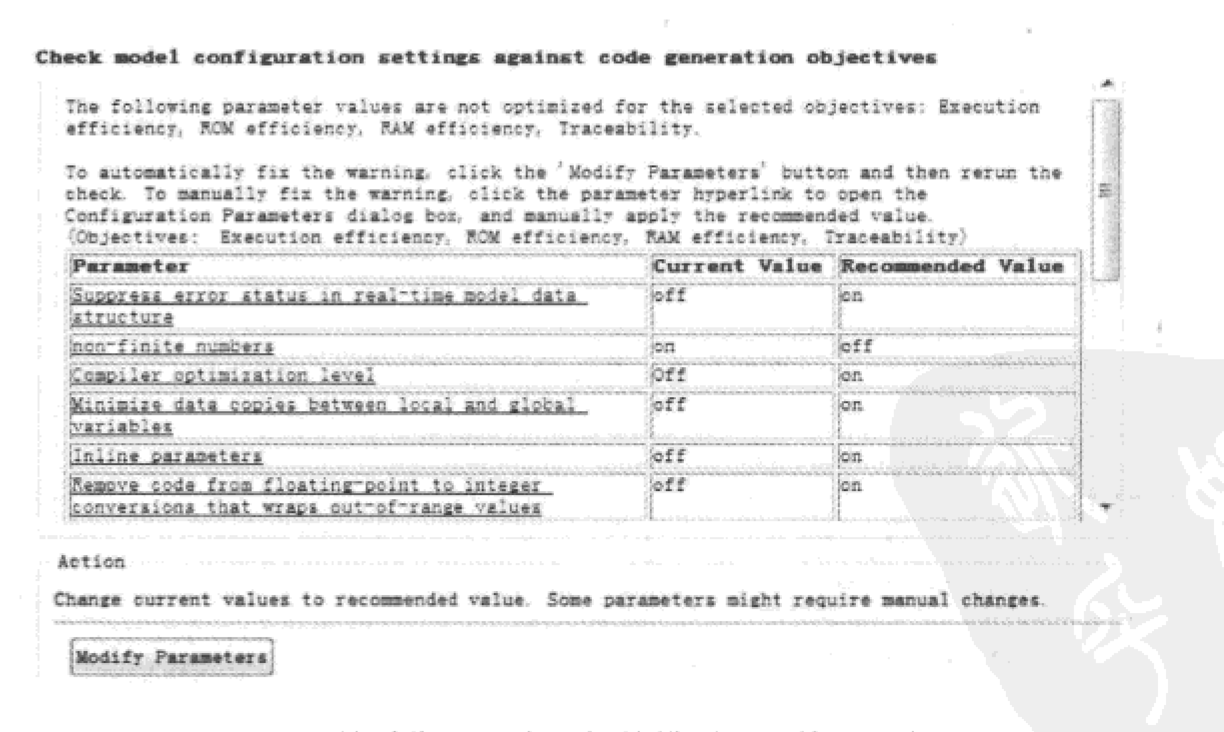
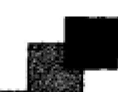
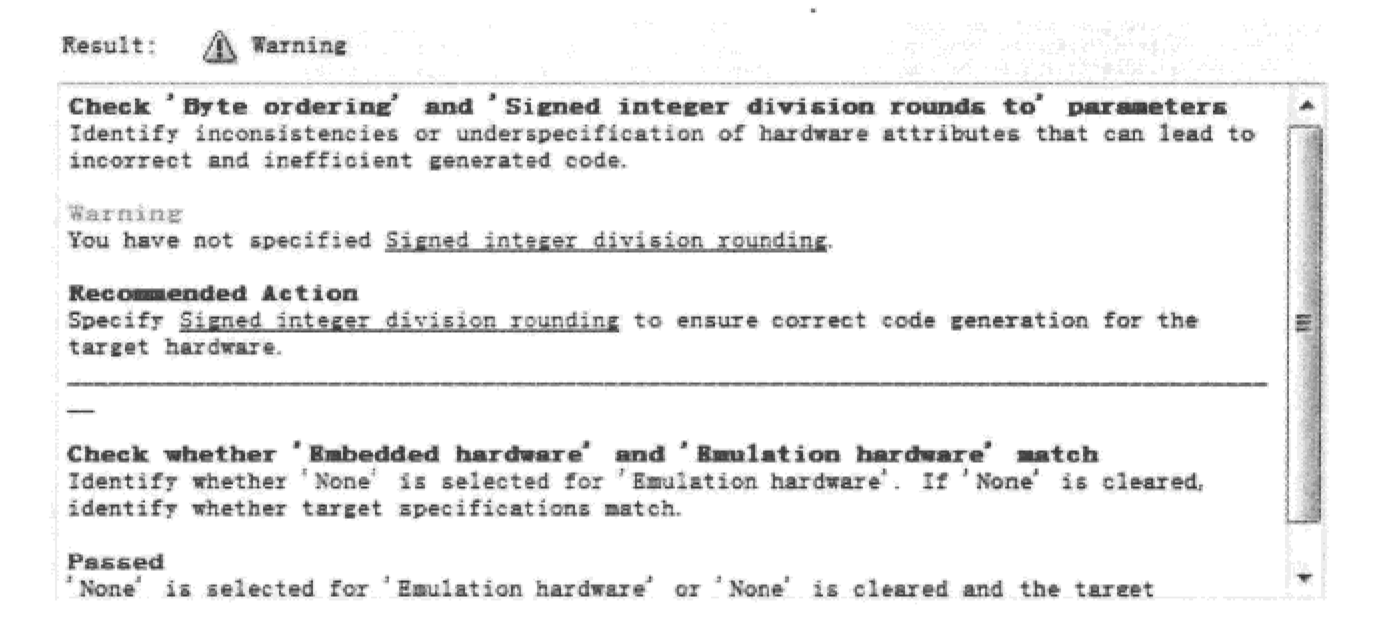


图 9.9.11　针对代码生成目标的模型设置检查任务页面

(5) 修改后右击在顶层目录 Code Generation Objectives 选择 Run Selected Checks 命令，则与④相关的一些警告也一并解决了。在 Check the Hardware Implementation 警告项中提



示要指定取整方式，如图 9.9.12 所示。用户可单击链接 Signed integer division rounding，在参数设置的 Hardware Implementation 界面将取整方式设为 zero。



Result: Warning

Check 'Byte ordering' and 'Signed integer division rounds to' parameters
Identify inconsistencies or underspecification of hardware attributes that can lead to incorrect and inefficient generated code.

Warning
You have not specified Signed integer division rounding.

Recommended Action
Specify Signed integer division rounding to ensure correct code generation for the target hardware.

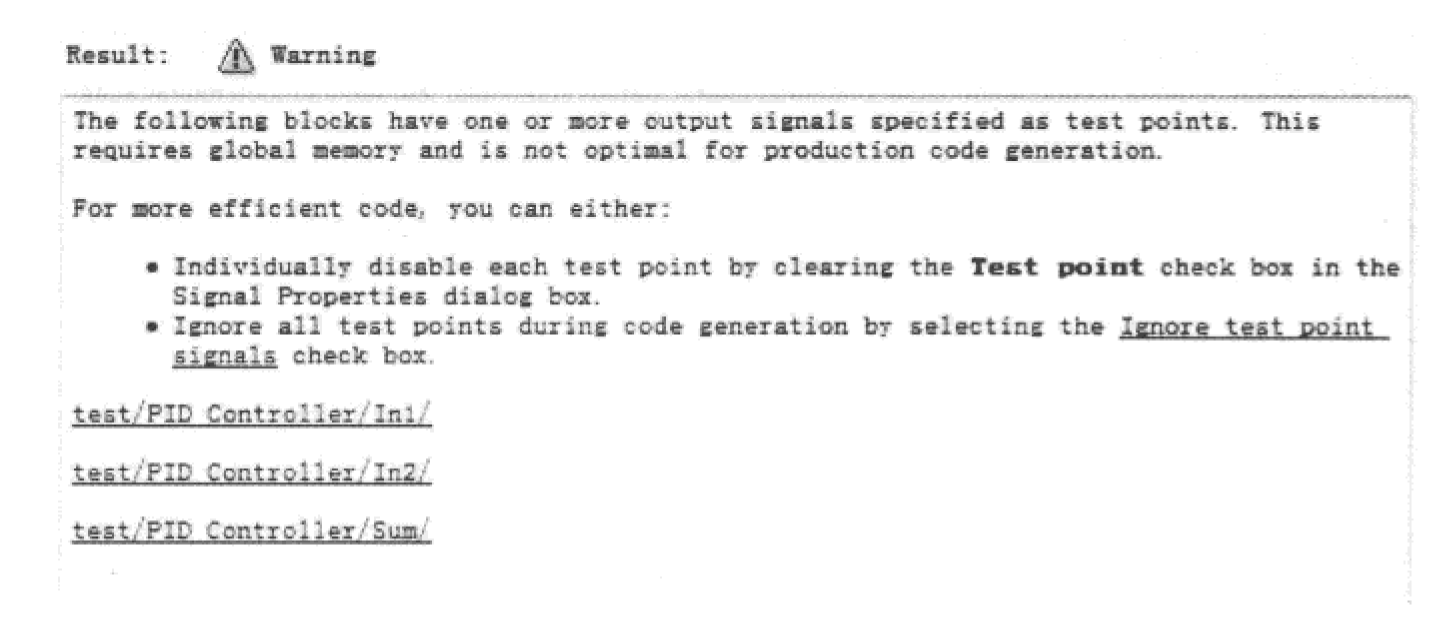
—

Check whether 'Embedded hardware' and 'Emulation hardware' match
Identify whether 'None' is selected for 'Emulation hardware'. If 'None' is cleared, identify whether target specifications match.

Passed
'None' is selected for 'Emulation hardware' or 'None' is cleared and the target

图 9.9.12　检查硬件设置任务界面

(6) Identify questionable code instrumentation 警告项中提示测试点会影响代码优化的效果，如图 9.9.13 所示。



Result: Warning

The following blocks have one or more output signals specified as test points. This requires global memory and is not optimal for production code generation.

For more efficient code, you can either:

- Individually disable each test point by clearing the **Test point** check box in the Signal Properties dialog box.
- Ignore all test points during code generation by selecting the Ignore test point signals check box.

test/PID Controller/In1/

test/PID Controller/In2/

test/PID Controller/Sum/

图 9.9.13　存疑的代码工具任务界面

用户可单击链接 test/PID Contoller/In1，定位到含测试点的信号线，在右键菜单中选择 Signal Properties 命令，如图 9.9.14 所示，取消勾选 test point 复选框。In2、Sum 的信号线也做同样处理。

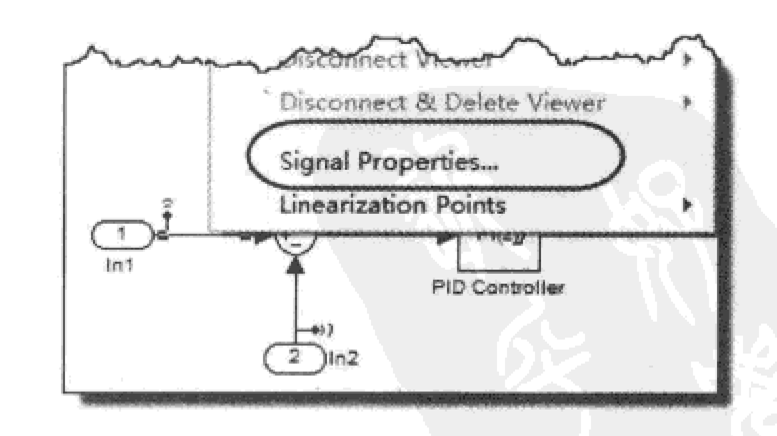


图 9.9.14　设置信号属性

(7) Identify blocks that generate expensive saturation and rounding codes 警告项提示说，当前设置会在代码中产生与整数除法相关的冗余，但可以起到保护算法中可能出现的分母为 0 等意外状况，如图 9.9.15 所示。用户可根据自己的需要判断是否需要这些冗余的保护措施。

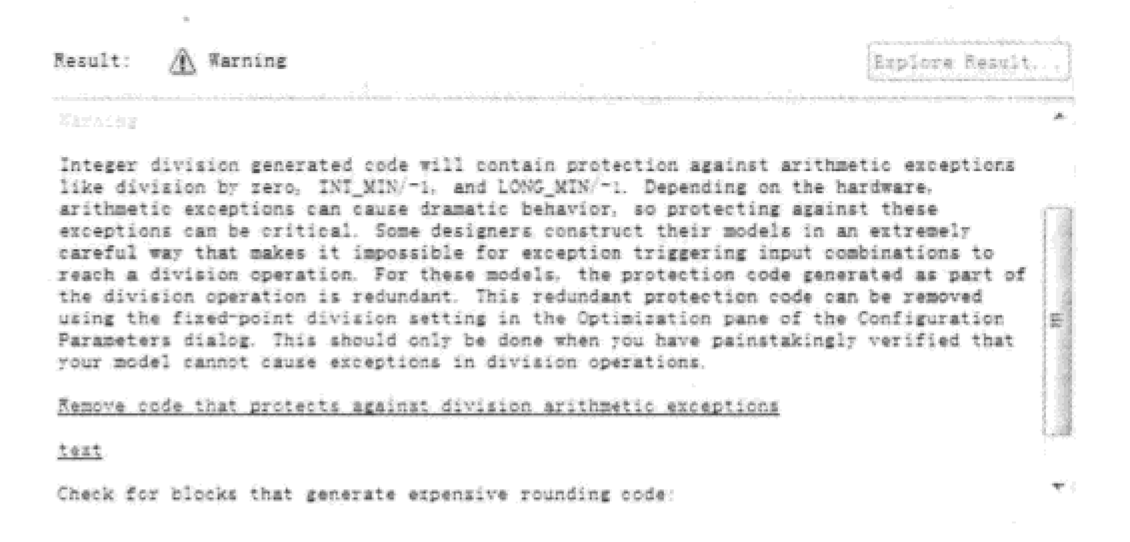


图 9.9.15 确认生成大量饱和、取整代码的模块任务界面

(8) Identify questiona fixed-point operations 警告项提示说定点数的乘法应遵循一定的规则，如图 9.9.16 所示。例如两个 32 位定点数相乘的结果只有用 64 位定点数表示才能保证不发生溢出错误，根据 9.5 节“定点模型”得出的结论，输出的结果仍然用 32 位表示，因此该项检查认为可能会出现问题。而实际上，在 Fixed Point Tool 中已经验证了 32 位定点数在本模型中是不会产生溢出的，所以这项警告可不用理会。

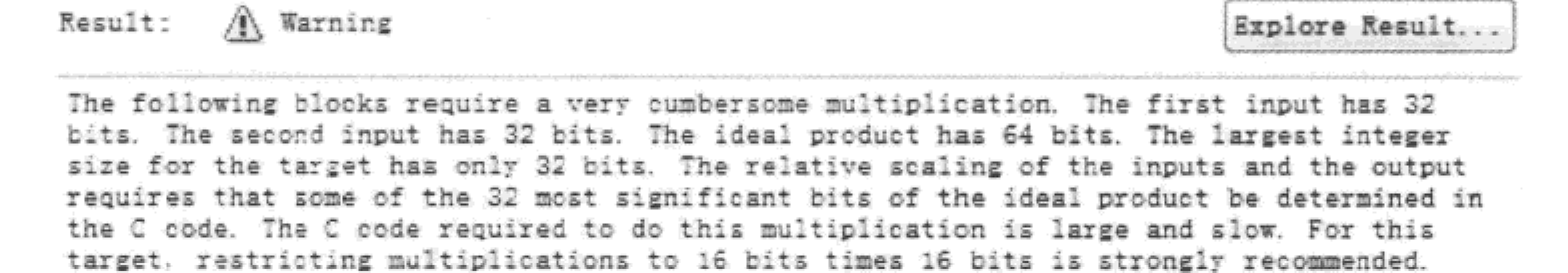


图 9.9.16 确认存疑的定点化处理任务界面

9.9.4 代码生成

在代码生成之前，需要考虑到在 ARM 上实现，将输入端口的数据类型设为 int32，输出端口的数据类型设为 uint32。相应地，需要添加数据类型转换模块使之与 PID Controller 子系统相匹配，如图 9.9.17 所示。

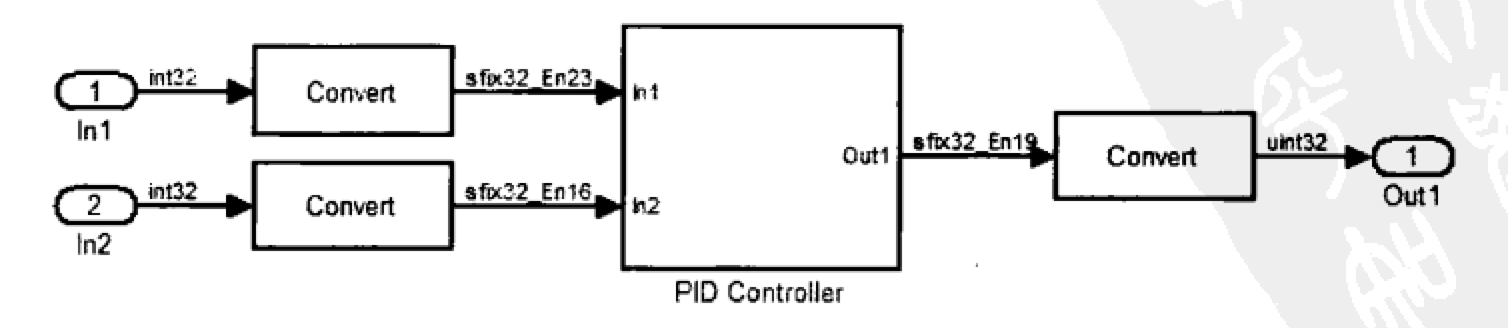
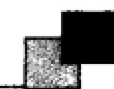


图 9.9.17 代码模型



在参数设置界面的 Hardware Implementation 界面选择 ARM7 后，单击按钮 生成代码，并自动弹出报告，如图 9.9.18 所示。

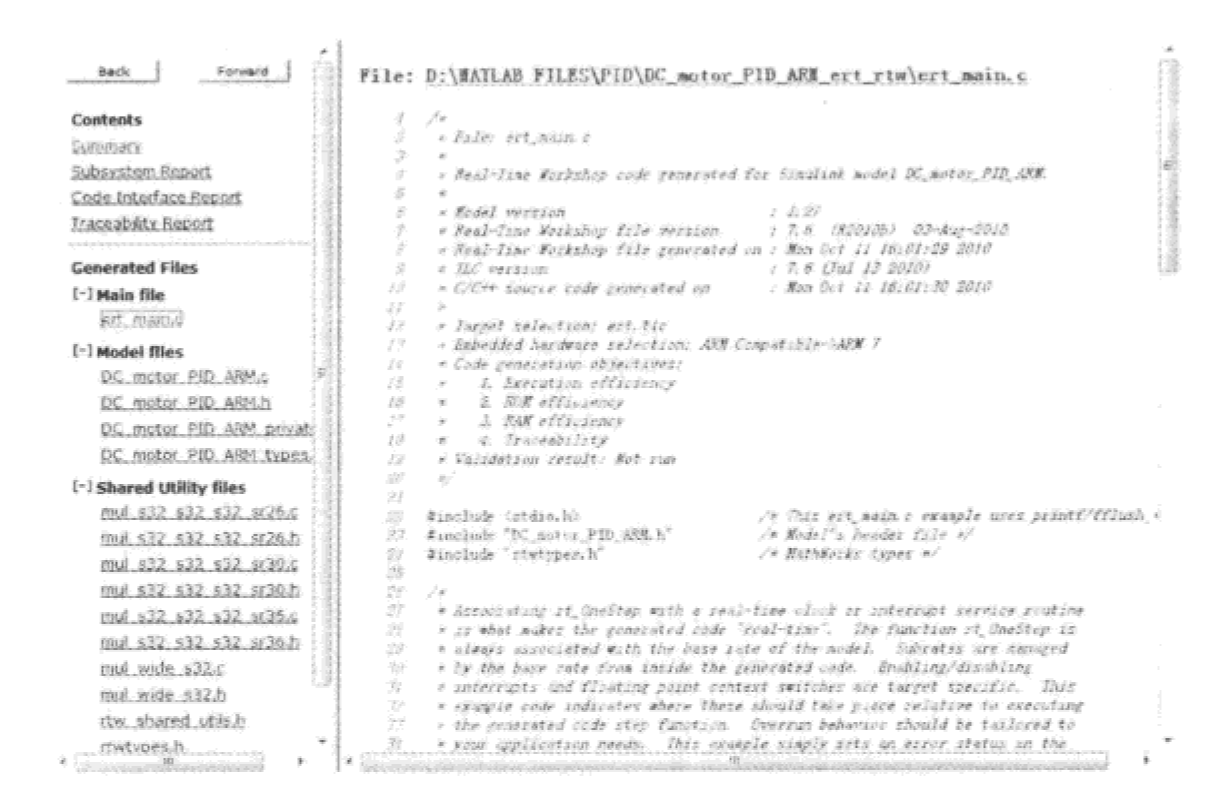


图 9.9.18　代码生成报告

9.10　虚拟硬件测试

在 Proteus ISIS 中绘制图 9.10.1 所示的原理图。

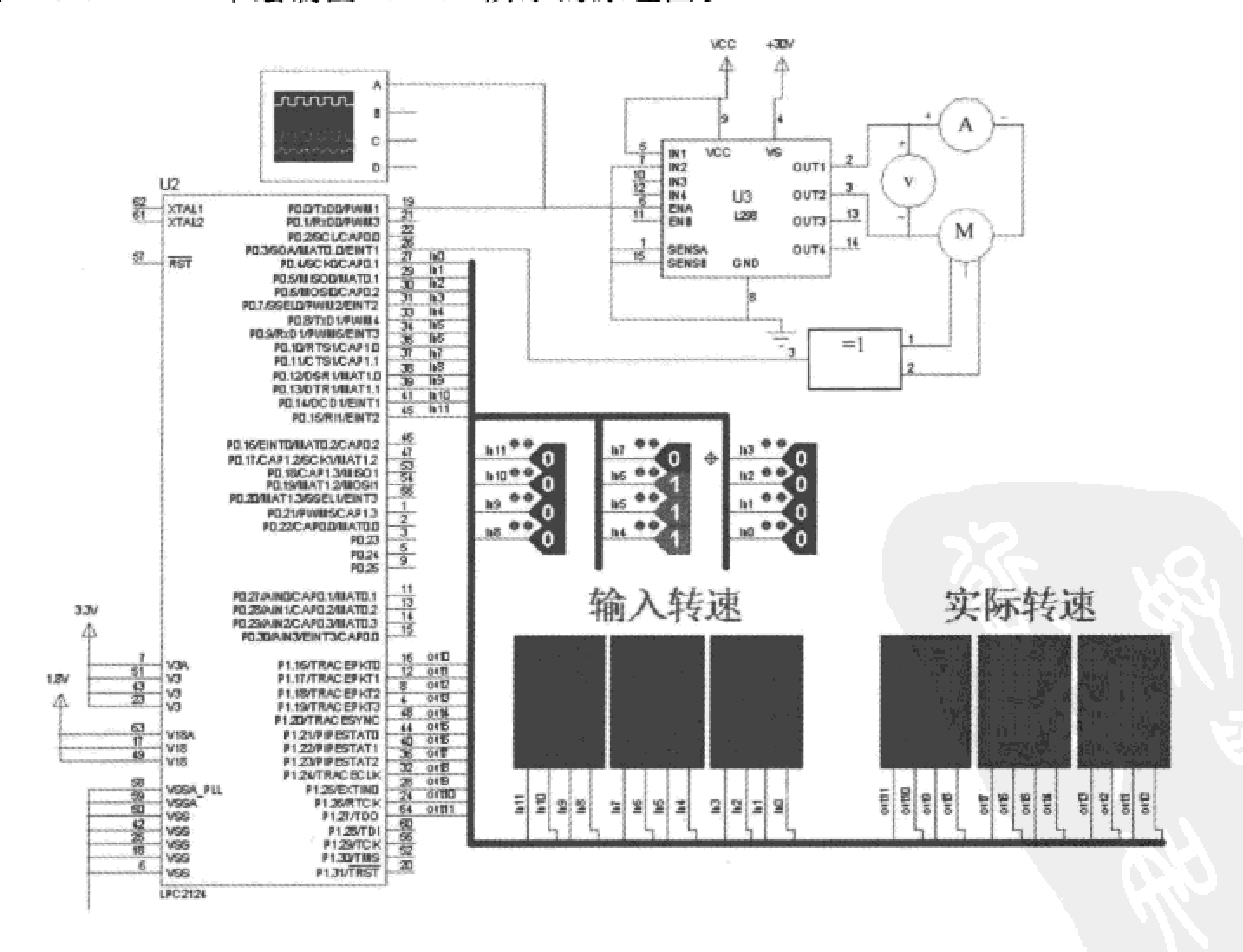


图 9.10.1　Proteus 原理图

该原理图采用 LPC2124 芯片实现对编码电动机的 PID 控制。L298 用于驱动电动机；74LS386 采集电动机的输出脉冲，从而得到电动机的实际转速；P0.0 引脚输出 PWM 控制信号，可通过示波器观察输出波形；Logic State 用来设定期望的转速，并由处于上方的数码管显示期望的转速；下方的数码管用以显示电动机的实际转速。

需要说明的是：PID 控制器的参数调节与具体的被控对象有关，本章的 PI 参数是根据第 2 章建立的直流电动机模型确定的，在实际应用中，若是用该算法控制一台与模型响应特性相符的实际电动机，可以达到很好的控制效果。

由于条件所限，作者手中并没有实际电动机，只能在 Proteus 中虚拟验证，而 Proteus 中电动机的建模理论和参数设置作者也没有找到更多可用的资料，无法针对 Proteus 中的电动机建立一个正确的 Simulink 模型。因此，基于第 2 章的电动机模型调节出来的 PI 参数应用到 Proteus 电动机中，效果肯定不理想。

为了解决此问题，一个折中的方法是利用经验，手动调节出一个合适的 PI 参数，然后在 Proteus 中验证。

1. 修改模型参数

在 PID 模块中手动将 PI 参数值设置为 30、0.08，如图 9.10.2 所示。

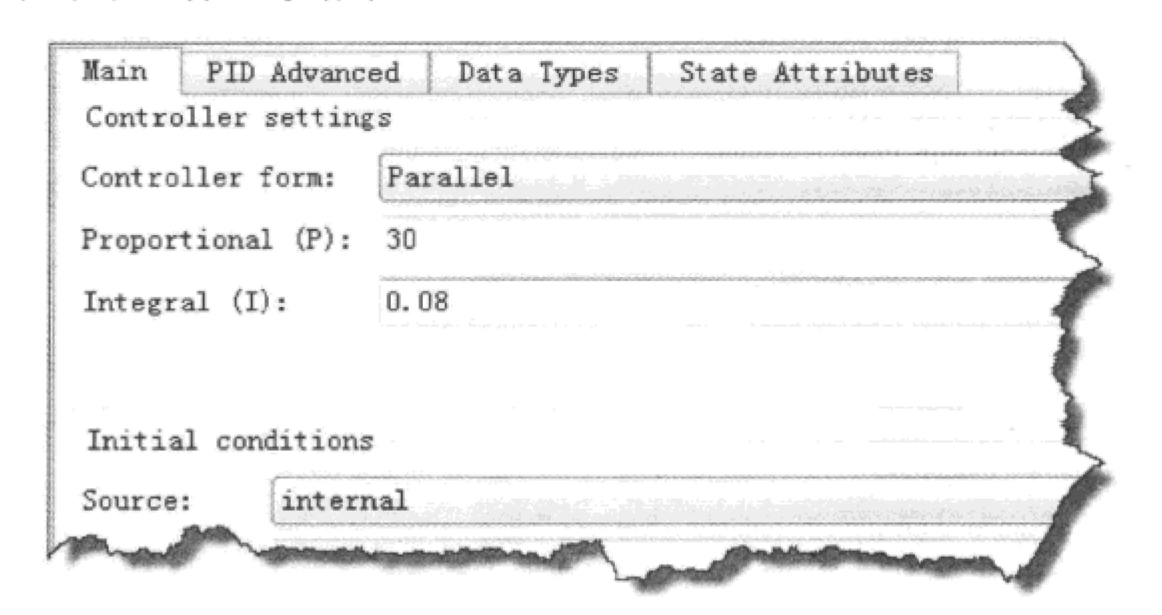


图 9.10.2　设置 P、I 参数

然后按照本章介绍的方法对其作验证、定点化处理等一系列基于模型设计的流程。

最后，在 PID 模块的 PID Advanced 界面为 PID 输出添加上下限，如图 9.10.3 所示。

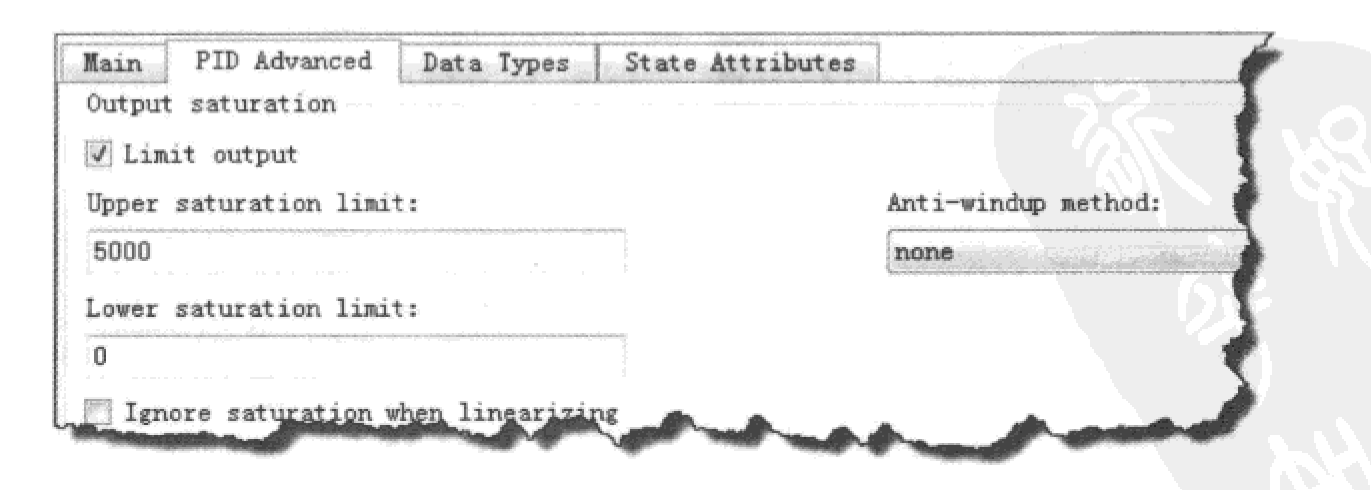
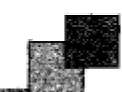


图 9.10.3　设置 PID 模块输出上下限



2. 生成代码并建立 Keil 工程

修改过参数并经过一系列优化、验证后，即可生成代码了。在 Keil 中选择 LPC2124 芯片，建立工程并将生成的代码添加进工程，如图 9.10.4 所示。

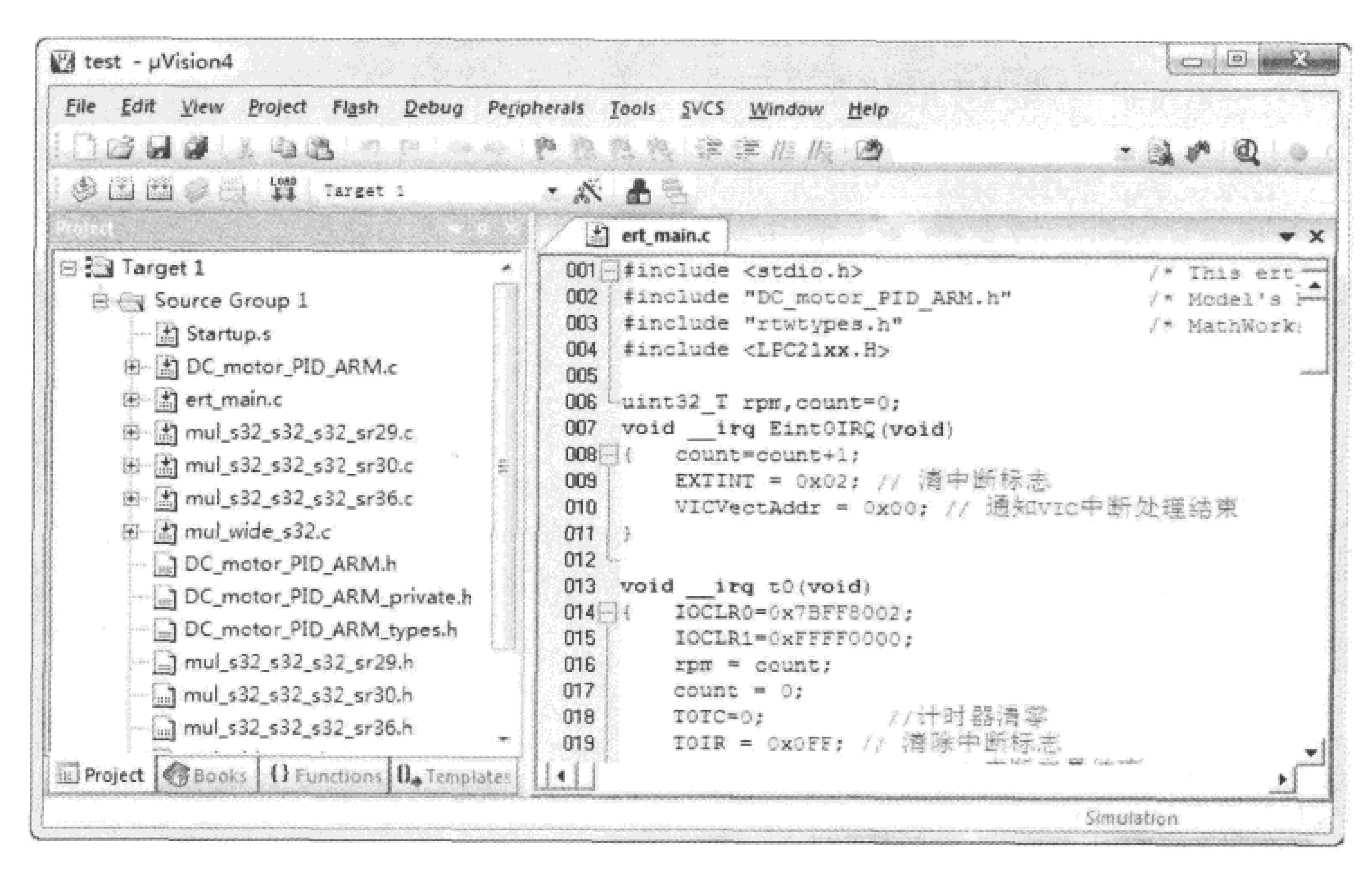


图 9.10.4　Keil 工程

由于生成的代码仅能完成 PID 控制算法，用户还需要添加芯片的底层驱动程序，才能完成整个嵌入式应用。打开 ert. main. c 文件，作以下修改：

```
//#include <stdio.h>                /* 删去该行 */
#include "DC_motor_PID_ARM.h"          /* 模型头文件 */
#include "rtwtypes.h"             /* MathWorks 数据类型 */
#include <LPC21xx.H>

uint32_T rpm,count = 0;
void __irq Eint0IRQ(void)       //外部中断服务程序
{     count = count + 1;
      EXTINT  =  0x02; // 清除中断标志
      VICVectAddr  =  0x00; // 通知 VIC 中断处理结束
}

void __irq t0(void)           //定时器 0 中断服务程序
{     IOCLR0 = 0x7BFF8002;
      IOCLR1 = 0xFFFF0000;
      rpm  =  count;
      count  =  0;
```



```
        T0TC = 0;          //计时器清零
        T0IR = 0x0FF; // 清除中断标志
        VICVectAddr = 0x00; // 中断向量结束
}

void Timer0(void)  //定时器 0 初始化
{       T0TCR = 0x02;  // 定时器 0 复位
        T0PR = 99;  //100 分频
        T0MCR = 0x03;  // 匹配后复位 TC,并产生中断
        T0MR0 = 12500; // 0.1 s 匹配值,OSC = 12.5M = 12 500 000
        T0IR = 0xFF;  // 清除中断标志
        T0TCR = 0x01;  // 启动定时器 0

        VICIntSelect = VICIntSelect & (~(1 << 4)); // 定时器 0 分配为 IRQ 中断
        VICVectCntl0 = 0x20 | 4; // 定时器 0 分配为向量 IRQ 通道 0
        VICVectAddr0 = (uint32_T)t0; // 分配中断服务程序地址
        VICIntEnable = 1 << 4; // 定时器 0 中断使能
}

void PWM(void)          //PWM 初始化
{       PWMPR = 99;          // 分频 120
        PWMMCR = 0x00000002;// MR0 匹配 TC 时复位 000 000 000 000 000 000 010
        PWMMR0 = 5000;
        PWMMR1 = 200;
        PWMLER = 0x7F;
        PWMPCR = 0x200;  // 使能 PWM1
        PWMTCR = 0x09;  // 使能 PWM 并开始计时
}

void ExInt(void)     //外部中断
{       EXTMODE = 0x02; // 设置外部中断为低电平触发
        EXTPOLAR = 0x02;
        VICIntSelect = 0 << 15; // 选择 EINT0 为 FIQ 中断
        VICVectCntl12 = 0x20 | 15; // 将外部中断 0 分配给向量中断 0
        VICVectAddr12 = (uint32_T)Eint0IRQ; // 设置中断服务程序地址
        VICIntEnable = 1 << 15; // 使能 EINT0 中断
        EXTINT = 0x02; // 清除 EINT0 中断标志
}

uint32_T input,input1,input2,input3;
void input_PWM(void)          //用模型算法控制 PWM 脉宽
{       input1 = (IOPIN0&0x4)>> 2|(IOPIN0&0x70)>> 3;
        input2 = (IOPIN0&0x780)>> 7;
```



```
        input3 = (IOPIN0&0x7800)>>11;
        PWMMR0 = 1000;
        PWMMR1 = (input2×10 + input1)×10;
        PWMLER = 0x7F;  // 使能 PWM0-PWM6 匹配锁存
}
uint32_T in;
void rt_OneStep(void);
void rt_OneStep(void)
{   in = (IOPIN0&0xFFF0)>>4;
    DC_motor_PID_ARM_U.In1 = in;
    DC_motor_PID_ARM_U.In2 = rpm;
    DC_motor_PID_ARM_step();
    PWMMR0 = 1000;
    //PWMMR1 = 999;
    PWMMR1 = (DC_motor_PID_ARM_Y.Out1);
    PWMLER = 0x7F;
}

int_T main(int_T argc, const char_T *argv[]);
int_T main(int_T argc, const char_T *argv[])
{
    DC_motor_PID_ARM_initialize();
    PINSEL0 = 0x000000C2;
    PINSEL1 = 0x00000000; //
    PINSEL2 = PINSEL2&0xFFFFFFF3;//P1.16 - 31 = GPIO
    IODIR0 = 0xFFFF0001; //P0.16 - 31 出 1,P0.1 - 0.15 入 0,P0.0 出 1
    IODIR1 = 0xFFFF0000; //P1.16 - 1.31 出 1
    PWM();
    Timer0();
    ExInt();
    while(1)
    {
        rt_OneStep();

        IOSET1 = rpm<<16;
}
    }
```

单击 Keil 工具栏的按钮，或按 Alt+F7 组合键，在 Output 选项卡，点选 Create HEX File 单选按钮，如图 9.10.5 所示。

再单击工具栏按钮，重编译工程(图 9.10.6)，窗口下部的信息显示已成功生成 HEX 文件。

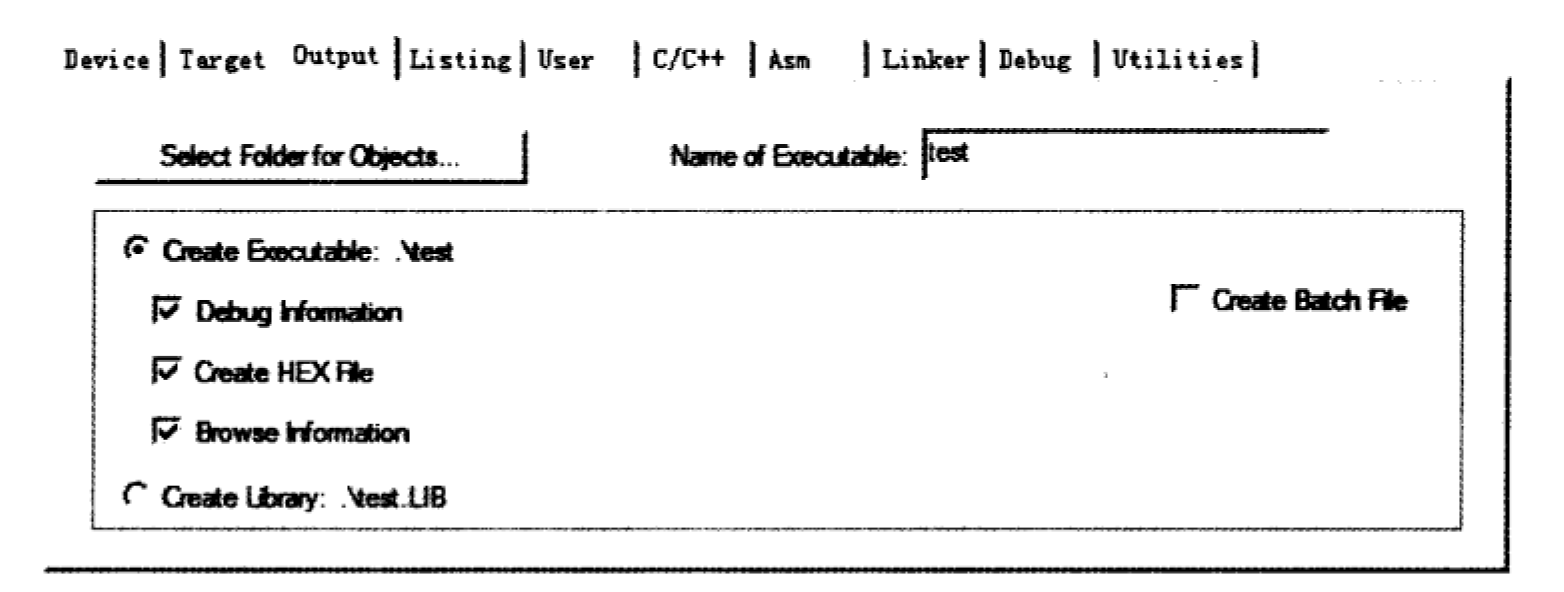


图 9.10.5　设置输出 HEX 文件

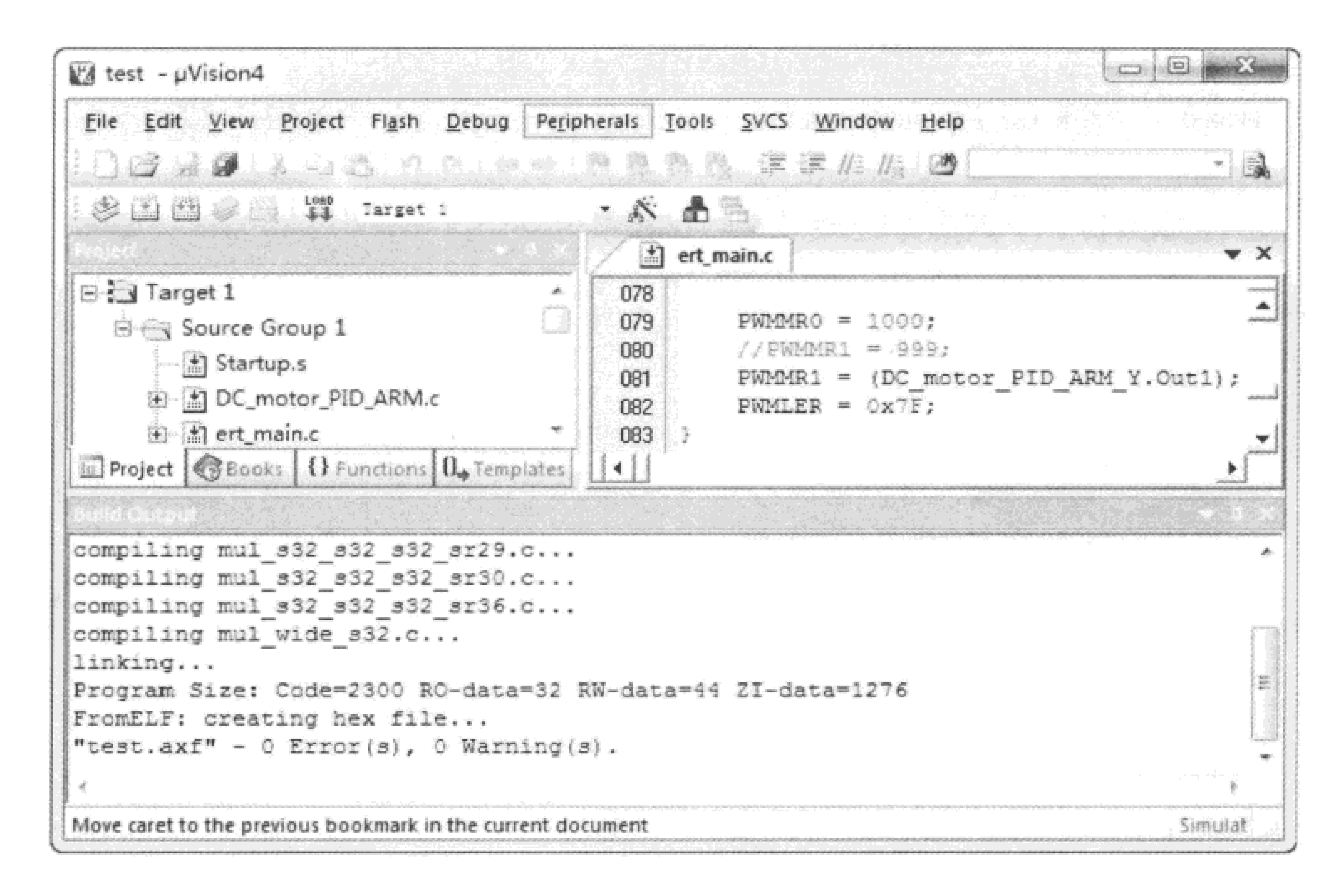
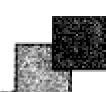


图 9.10.6　编译信息

3. 加载 HEX 文件,观察实验结果

加载先前生成的 HEX 文件,按下仿真按钮并调节 Logic State,设置期望转速,如图9.10.7所示。

从代码层面看,Keil 工程中混合使用了手写代码与自动生成代码,其中自动生成的代码 400 多行,手写代码 100 行左右。自动生成的代码占到了总代码量的 80%,显著提高了系统开发效率,降低了开发成本。一般来说,在基于模型设计中需要手写的代码大多为硬件底层驱动代码,因此,在大型项目中,系统的功能与算法的部分代码更加复杂,而硬件底层驱动则不会有明显变化,这样一来,自动生成代码在总代码量中的比例将会更高,更加凸显出基于模型设计高效率、低成本的优势。现在已经有部分硬件厂商发现了基于模型设计的优点,并为自己的产品编写了驱动模块。例如使用 Microchip 公司为 dsPIC 芯片开发的 dsPIC Blocksets,可以自



动生成驱动代码，达到完全不需要手写代码的目的。

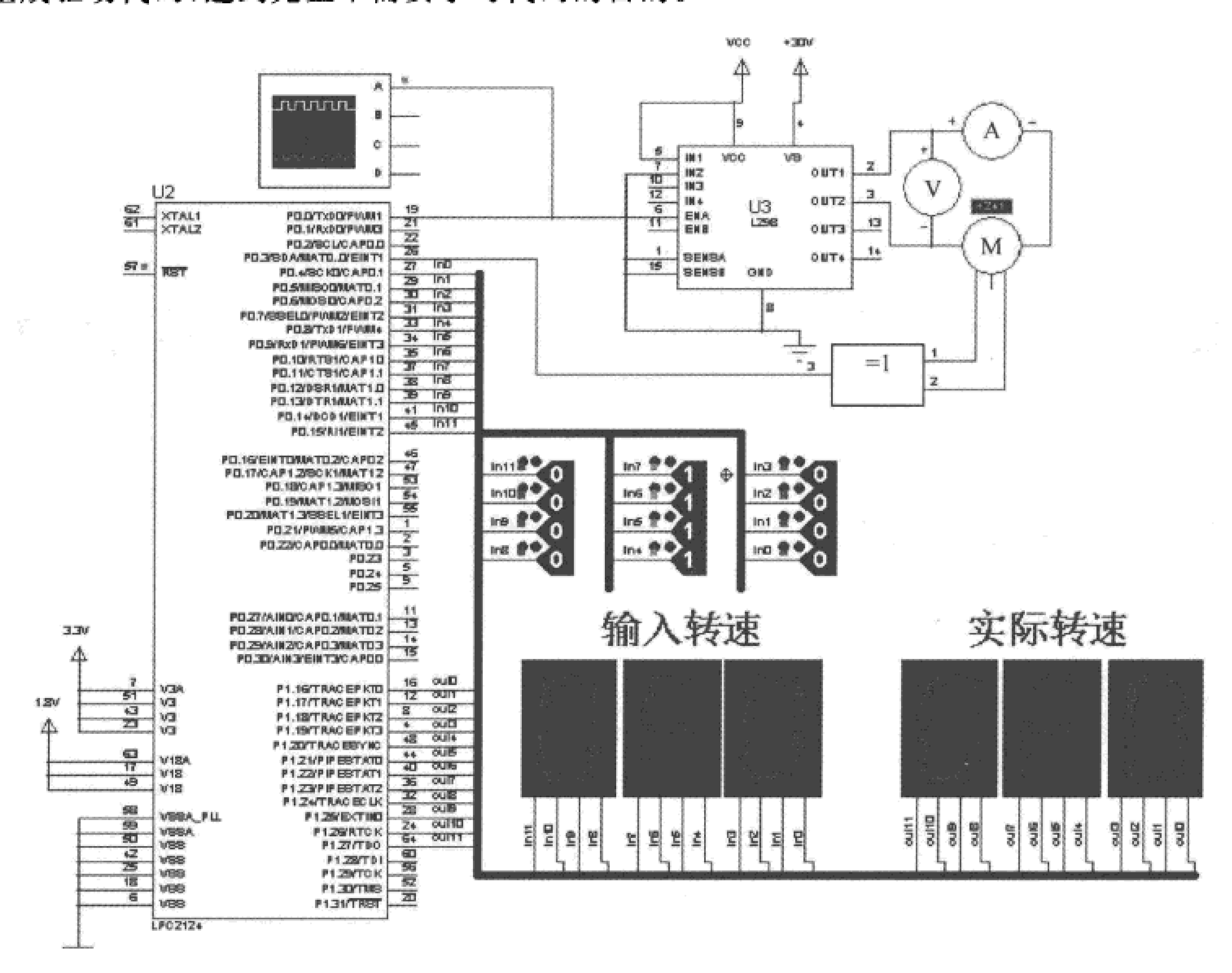


图 9.10.7 Proteus 仿真结果

从虚拟硬件测试结果看，电动机实际转速与控制信号几乎完全一致，也和本章 9.4 节介绍的功能验证模型、9.7 节的 SIL 测试模型的功能一致。这充分证明了手工代码加自动生成代码混合使用，有效地提高了开发效率，实现了基于模型设计所倡导的从概念到实现的设计理念。



附录

Embedded MATLAB 支持的各函数

1. 航空航天模块集函数

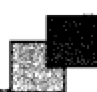
函数名	描　述	函数名	描　述
quatconj	计算共轭四元数	quatmultiply	计算两个四元数的积
quatdivide	一个四元数除以另一个四元数	quatnorm	计算四元数的范数
quatinv	计算四元数的逆	quatnormalize	标准化四元数
quatmod	计算四元数的模		

2. 算术运算符函数

函数名	描　述	函数名	描　述
ctranspose	复数转置	idivide	整型数据相除并取整
isa	确定当前的对象是否为指定的数据类型	ldivide	数组左除
minus	减	mldivide	矩阵左除
mpower	数组幂运算	mrdivide	矩阵右除
mtimes	矩阵乘	plus	加
power	数组幂运算	rdivide	数组右除
times	数组乘法	transpose	矩阵转置
uminus	一元减	uplus	一元加

3. 位操作函数

函数名	描　述
swapbytes	交换变量高低位字节排列顺序



4. 强制数据类型转换函数

数据类型	描　述	数据类型	描　述
cast	变量转换为另一种数据类型	char	建立字符(串)阵列
class	查询对象的类型	double	转换为双精度浮点数
int8,int16,int32	转换为有符号整型数据	logical	转换为布尔型数据
single	转换为单精度浮点数	typecast	转换数据类型(不改变基础数据)
uint8,uint16,uint32	转换为无符号整型数据		

5. 通信工具箱函数

函数名	描　述	函数名	描　述
bi2de	二进制向量转成十进制	de2bi	十进制向量转成二进制
istrellis	判断是否为正确的栅格图	Poly2trellis	将卷积码多项式转化为栅格图
rcosfir	升余弦 FIR 滤波器		

6. 复数运算函数

函数名	描　述	函数名	描　述
complex	用实部和虚部构造复数	conj	返回复数的共轭
imag	返回复数的虚部	isnumeric	若是数值数组,则返回 1
isreal	若是复数,则返回 0	isscalar	若是标量数组,则返回 1
real	返回复数的实部	unwrap	校正相位角,得到平滑的相位曲线

7. 数据类型函数

函数名	描　述	函数名	描　述
iscell	确定输入是否为单元阵列	nargchk	验证输入参数值
nargoutchk	验证输出参数值		

8. 导数与积分函数

函数名	描　述	函数名	描　述
cumtrapz	累积梯形数值积分	diff	差分、近似导数
trapz	梯形数值积分		

9. 离散数学函数

函数名	描　述	函数名	描　述
lcm	最小公倍数	gcd	最大公约数
nchoosek	计算组合数		

10. 错误处理函数

函数名	描　述	函数名	描　述
assert	当运行环境被破坏时,报告错误	error	显示错误信息,并终止函数

11. 指数函数

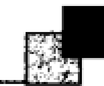
函数名	描　述	函数名	描　述
exp	以e为底的指数函数	expm	矩阵指数函数
expm1	计算 $\exp(x)-1$	factorial	阶乘
log	自然对数	log2	以2为底的对数
log10	以10为底的对数	log1p	计算 $\lg(1+x)$
nextpow2	以2为底的对数,并向上取整	nthroot	实数根
reallog	非负实数的自然对数	realpow	非负实数的幂
realsqrt	非负实数的平方根	sqrt	平方根

12. 滤波和卷积函数

函数名	描　述	函数名	描　述
conv	卷积或多项式乘法	conv2	二维卷积
deconv	反卷积或多项式除法	detrend	去除矩阵中的线性趋势
filter	一维数字滤波	filter2	二维数字滤波

13. 定点工具箱函数

函数名	描　述
abs	求绝对值
add	两个对象相加(fimath 对象)
all	确定数组元素是否都为非0
any	确定数组是否有非0元素

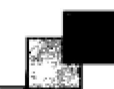


续 表

函数名	描　述
bitand	位求与
bitandreduce	连续位间相与
bitcmp	位取反
bitconcat	合并字符串
bitget	取特定位
bitor	位求或
bitorreduce	连续位间相或
bitreplicate	复制合并字符串
bitrol	位左移
bitror	位右移
bitset	特定位置 1
bitshift	位移动
bitsliceget	取连续位部分
bitsll	位逻辑左移
bitsra	位算术右移
bitsrl	位逻辑右移
bitxor	位异或
bitxorreduce	连续位间相异或
ceil	沿＋∞方向取整
complex	用实部、虚部构造复数对象
conj	共轭复数函数
conv	卷积或多项式乘法
convergent	向最接近的偶整数取整
cordiccexp	基于 CORDIC 的复指数近似
cordiccos	基于 CORDIC 的余弦近似
cordicsin	基于 CORDIC 的正弦近似
cordicsincos	基于 CORDIC 的正弦与余弦近似
ctranspose	复共轭转置
diag	建立对角矩阵或获取对角向量
disp	显示矩阵与文本
divide	对象相除
double	双精度浮点值(有意义的值)
end	数组的最后一个下标
eps	量化对象的相对量化精度
eq	相等(有意义的值)
fi	构造 fi 对象

续 表

函数名	描　述
filter	fi 对象的一维滤波器
fimath	构造 fimath 对象
fix	沿零方向取整
floor	沿 $-\infty$ 方向取整
ge	大于等于(有意义的值)
get	获得对象属性
getlsb	最低有效位
getmsb	最高有效位
gt	大于(有意义的值)
horzcat	横向合并多重对象
imag	求虚部函数
int8,int16, int32	内置 8、16、32 位整形数据存储对象的整数值
iscolumn	判断对象是否为列向量
isempty	判断数组是否为空
isequal	判断是否相等(有意义的值)
isfi	判断变量是否为 fi 对象
isfimath	判断变量是否为 fimath 对象
isfimathlocal	判断 fi 对象是否附加有 fimath 对象
isfinite	检查数组中的有限元素
isinf	检查数组中的无限元素
isnan	判断数组元素是否为 NAN
isnumeric	判断输入是否为数值数组
isnumerictype	判断变量是否为数值型对象
isreal	判断数组元素是否为实数
isrow	判断对象是否为行向量
isscalar	判断输入是否为标量
issigned	判断对象是否有符号
isvector	判断输入是否为向量
le	小于等于(有意义的值)
length	查询向量的维数
logical	将数字量转化为逻辑量
lowerbound	fi 对象的下界范围
lsb	fi 对象最低有效位的缩放
lt	小于(有意义的值)
max	fi 对象数组中的最大元素
mean	定点矩阵的均值



续 表

函数名	描 述
median	定点矩阵的中值
min	fi 对象数组中的最小元素
minus	fi 对象矩阵间的差值
mpower	定点矩阵求幂
mpy	两个对象相乘(用 fimath 对象)
mrdivide	矩阵右除
mtimes	fi 对象矩阵乘积
ndims	求矩阵维数
ne	不等于(有意义的值)
nearest	向最接近的正整数取整
numberofelement	fi 数组中数据元素的个数
numerictype	构造数值型对象
permute	任意改变矩阵维数序列
plus	fi 对象矩阵相加
pow2	基 2 标量浮点数
power	定点矩阵中对各元素求幂
range	fi 或量化对象的数值范围
rdivide	数组右除
real	求实部函数
realmax	最大浮点数值
realmin	最小浮点数值
reinterpretcast	转换定点数据类型(不改变基础数据)
repmat	复制并排列矩阵函数
rescale	fi 对象缩放
reshape	矩阵维度变换
round	舍入取整
sfi	构造有符号定点数值型对象
sign	符号函数
single	fi 对象的(有意义的)单精度浮点值
size	查询矩阵的维数
sort	对向量中各元素排序
sqrt	平方根函数
sub	两个对象相减(用 fimath 对象)
subsasgn	下标分配
subsref	下标引用
sum	对向量中各元素求和

续 表

函数名	描　述
times	fi 对象的元素间相乘
transpose	转置
tril	取矩阵的下三角部分
triu	取矩阵的上三角部分
ufi	构造无符号定点数值型对象
uint8、uint16、uint32	内置 8、16、32 位无符号整型数据存储对象的整数值
uminus	fi 对象数组取反
uplus	一元加
upperbound	fi 对象的上界范围
vertcat	纵向合并多重 fi 对象

14. 直方图函数

函数名	描　述	函数名	描　述
hist	直方图绘制	histc	直方图计数

15. 图像处理工具箱函数

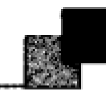
函数名	描　述	函数名	描　述
fspecial	创建预定义的二维滤波器	label2rgb	将 labelmatrix 换为 RGB 图像

16. 输入输出函数

函数名	描　述	函数名	描　述
nargin	函数中参数输入个数	nargout	函数中输出变量个数

17. 插值和几何运算

函数名	描　述	函数名	描　述
cart2pol	笛卡儿坐标到极坐标转换	cart2sph	笛卡儿坐标到球面坐标转换
interp1	一维线性插值	interp1q	快速一维线性插值
meshgrid	构造三维图形用 x,y 阵列	pol2cart	极坐标到笛卡儿坐标转换
sph2cart	球面坐标到笛卡儿坐标转换		



18. 线性代数

函数名	描　述	函数名	描　述
linsolve	求解线性系统方程	rsf2csf	将实 Schur 转化为复 Schur
schur	Schur 分解	sqrtm	矩阵平方根

19. 逻辑运算函数

函数名	描　述	函数名	描　述
and	逻辑与	bitand	位求与
bitcmp	位取反	bitget	读取特定数据位
bitor	位求或	bitset	特定位置 1
bitshift	位平移	bitxor	位异或
not	逻辑反	or	逻辑或
xor	逻辑异或		

20. 矩阵和数组函数

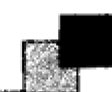
函数名	描　述	函数名	描　述
abs	取数组的绝对值或复数的模	all	检测所有元素是否为非零
angle	角相位函数	any	测试向量中是否有真元素
bsxfun	两个矩阵对应元素之间的运算	cat	向量连接
circshift	数组循环移位	compan	生成伴随矩阵
cond	求矩阵的条件数	cov	协方差计算
cross	向量叉积	cumprod	向量累积
cumsum	向量累加	det	求矩阵的行列式
diag	建立对角矩阵，或获取矩阵对角线	diff	差分与近似微分
dot	向量点积	eig	求矩阵的特征值和特征向量
eye	产生单位矩阵	false	生成一个指定大小的全零矩阵
find	查找非零元素下标	flipdim	按指定尺寸翻转数组
fliplr	左右翻转矩阵元素	flipud	上下翻转矩阵元素
full	稀疏矩阵转换为常规矩阵	hankel	hankel 矩阵
hilb	生成希尔伯特矩阵	ind2sub	将线性索引转换为下标
inv	求方阵的逆矩阵	invhilb	求希尔伯特矩阵的逆
ipermute	数组维数翻转	iscolumn	判断输入是否是列向量
isempty	判断数组是否全空	isequal	判断数组是否相等

续 表

函数名	描　述	函数名	描　述
isequalwithqualnans	判断数组是否相等，把 NANS 看做相等	isfinite	检查数组的有限元素
isfloat	判断输入是否是浮点数组	isinf	检查数组的无限元素
isinteger	判断输入是否是整数数组	islogical	判读输入是否是符合逻辑的数组
ismatrix	判断输入是否是矩阵	isrow	判断输入是否是行向量
isnan	检查数组中的 NAN 元素	issparse	判断输入是否是稀疏矩阵
isvetor	判断输入是否是向量	kron	张量积
length	测试矩阵长度	linspace	生成线性空间向量
logspace	生成对数空间向量	lu	LU 矩阵分解
magic	魔幻方阵	max	矩阵元素中的最大值
min	矩阵元素中的最小值	ndgrid	为 N－D 函数生成数组并插值
ndims	维度数	nnz	矩阵中非零元素个数
nonzeros	非零矩阵元素	norm	向量和矩阵的范数
normest	范数估计	numel	数组或下标数组的元素数
ones	建立一个全 1 矩阵	pascal	帕斯卡矩阵
permute	重新设置数组维数	pinv	矩阵的伪逆矩阵
planerot	特定平面旋转	prod	数组元素积
qr	直角三角转换	randperm	随机序列
rank	矩阵的秩	rcond	矩阵公约数估计
repmat	复制和覆盖一个数组	reshape	矩阵维数变换
rosser	经典对称特征值问题	rot90	90°旋转矩阵
shiftdim	改变维度	sign	符号函数
size	矩阵大小	sort	升序或降序排列元素
sortrows	升序排列行	squeeze	去除单一维度
sub2ind	将下标转换为线性索引	subspace	两个子空间的角度
sum	矩阵元素之和	toeplitz	托普利兹矩阵
trace	对角线元素之和	tril	取下三角部分
triu	取上三角部分	true	按指定尺寸生成一个逻辑数组
vander	范德蒙矩阵	wilkinson	威尔金森特征值测试矩阵
zeros	生成一个全零矩阵		

21. 非线性数值方法

函数名	描　述	函数名	描　述
fzero	求单变量连续函数的根	quad2d	平面域上二重积分的数值估计
quadgk	用自适应 Gauss－Kronrod 正交法求数值积分		



22. 多项式函数

函数名	描　述	函数名	描　述
poly	求矩阵的特征多项式	polyfit	数据的多项式拟合
polyval	多项式求值	roots	求多项式的根

23. 关系运算函数

函数名	描　述	函数名	描　述
eq	等于	ge	大于或等于
gt	大于	le	小于或等于
lt	小于	ne	不等于

24. 取整和取余函数

函数名	描　述	函数名	描　述
ceil	向正无穷方向取整	convergent	向最接近的偶整数取整
fix	沿零方向取整	floor	向负无穷方向取整
mod	取模(保留符号)	nearest	正无穷方向取整，并取最近的整数
rem	求除法的余数	round	舍入取整

25. 排序函数

MATLAB 支持以下的功能：

函数名	描　述	函数名	描　述
intersect	查找两个向量的交集	ismember	判断矩阵元素是否属于集合
issorted	判别集合中的元素是否有序	setdiff	查找两个向量的差集
setxor	查找两个向量的补集	union	查找两个向量的并集
unique	查找向量中所有不重复的		

26. MATLAB 信号处理函数

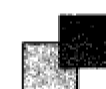
函数名	描　述	函数名	描　述
chol	Cholesky 分解	conv	卷给与多项式乘法
fft	离散傅里叶变换	fft2	2 维离散傅里叶变换

续 表

函数名	描　述	函数名	描　述
fftshift	fft 与 fft2 输出重排	filter	适用于实数和复数的数字滤波器过滤输入数据
freqspace	频率响应的频率间隔	ifft	离散傅里叶逆变换
ifft2	二维离散傅里叶逆变换	ifftshift	ifft 与 ifft2 输出重排
svd	奇异值分解	zp2tf	将零极点表示的滤波器改用传递函数表示

27. 信号处理工具箱函数

函数名	描　述
barthannwin	改进 Bartlett－Hann 窗(产生代码时需要信号处理模块集的许可)
bartlett	bartlett 窗
besselap	贝塞尔模拟低通滤波器原型
bitrevorder	按位倒序(产生代码时需要信号处理模块集的许可)
blackman	blackman 窗(产生代码时需要信号处理模块集的许可)
blackmanharris	最小 4－term Blackman－Harris 窗
bohmanwin	bohman 窗(产生代码时需要信号处理模块集的许可)
buttap	巴特沃斯滤波器原型
butter	巴特沃斯滤波器设计(产生代码时需要信号处理模块集的许可)
buttord	巴特沃斯滤波器的阶数和截止频率
cfirpm	复合非线性相位等波纹 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
cheb1ap	切比雪夫 I 型模拟低通滤波器原型
cheb2ap	切比雪夫 II 型模拟低通滤波器原型
cheb1ord	切比雪夫 I 型模拟低通滤波器阶数
cheb2ord	切比雪夫 II 型模拟低通滤波器阶数
chebwin	切比雪夫窗(产生代码时需要信号处理模块集的许可)
cheby1	切比雪夫 I 型滤波器设计(产生代码时需要信号处理模块集的许可)
cheby2	切比雪夫 II 型滤波器设计(产生代码时需要信号处理模块集的许可)
dct	离散余弦变换
downsample	降低采样速率
dpss	离散长球面序列(产生代码时需要信号处理模块集的许可)
ellip	椭圆滤波器设计(产生代码时需要信号处理模块集的许可)
ellipap	椭圆模拟低通滤波器原型
ellipord	椭圆滤波器的最小阶数
filtfilt	零相位数字滤波



续 表

函数名	描　述
fir1	窗函数有限冲激响应法滤波器设计(产生代码时需要信号处理模块集的许可)
fir2	频率抽样有限冲激响应法滤波器设计(产生代码时需要信号处理模块集的许可)
fircls	最小平方约束,FIR 多频带滤波器设计(产生代码时需要信号处理模块集的许可)
fircls1	最小平方约束,低通、高通、线性相位 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
firls	最小平方线性相位 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
firpm	Parks - McClellan 最佳 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
firpmord	Parks - McClellan 最佳 FIR 滤波器评价(产生代码时需要信号处理模块集的许可)
firrcos	升余弦滤波器设计(产生代码时需要信号处理模块集的许可)
flattopwin	平顶窗(产生代码时需要信号处理模块集的许可)
freqz	数字滤波的频率响应
gaussfir	高斯脉冲成型 FIR 滤波器(产生代码时需要信号处理模块集的许可)
gausswin	高斯窗(产生代码时需要信号处理模块集的许可)
hamming	汉明窗(产生代码时需要信号处理模块集的许可)
hann	汉宁窗(产生代码时需要信号处理模块集的许可)
idct	离散余弦逆变换
intfilt	内插 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
kaiser	凯瑟窗(产生代码时需要信号处理模块集的许可)
kaiserord	凯瑟窗 FIR 滤波器设计参数估计(产生代码时需要信号处理模块集的许可)
levinson	Levinson - Durbin 递归法
maxflat	一般数字巴特沃斯 FIR 滤波器设计(产生代码时需要信号处理模块集的许可)
nuttallwin	Nuttall 最小 4 - term Blackman - Harris 窗
parzenwin	Parzen 窗(产生代码时需要信号处理模块集的许可)
rectwin	矩形窗(产生代码时需要信号处理模块集的许可)
resample	调整采样率
sgolay	Savitzky - Golay 滤波器设计(产生代码时需要信号处理模块集的许可)
sosfilt	二价 IIR 滤波设计(产生代码时需要信号处理模块集的许可)
taylorwin	泰勒窗(产生代码时需要信号处理模块集的许可)
triang	三角窗(产生代码时需要信号处理模块集的许可)
tukeywin	tukey 窗(产生代码时需要信号处理模块集的许可)
upfirdn	增采样、然后 FIR 滤波、再减采样
upsample	提高采样速率
xcorr	互相关(产生代码时需要信号处理模块集的许可)
yulewalk	递归数字滤波器设计(产生代码时需要信号处理模块集的许可)

28. 特殊数值

符号	描　述	符号	描　述
eps	精度容许误差(无穷小)	inf	无穷大
intmax	特定数据类型的最大整数	intmin	特定数据类型的最小整数
NaN or nan	非数	pi	圆周率
rand	产生随机分布矩阵	randn	产生正态分布矩阵
realmax	最大浮点数值	realmin	最小浮点数值

29. 专用数学函数

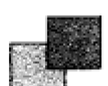
函数名	描　述	函数名	描　述
beta	beta 函数	betainc	非完全的 beta 函数
betaln	beta 对数函数	ellipke	完全椭圆积分
erf	误差函数	erfc	互补误差函数
erfcinv	逆补差函数	erfcx	比例互补误差函数
erfinv	逆误差函数	expint	指数积分函数
gamma	gamma 函数	gammainc	非完全 gamma 函数
gammaln	gamma 对数函数		

30. 统计函数

函数名	描　述	函数名	描　述
corrcoef	相关系数	mean	求向量中各元素均值
median	求向量中中间元素	mode	数组中出现频率最高的值
std	求向量中各元素标准差	var	方差

31. 字符串函数

函数名	描　述	函数名	描　述
bin2dec	二进制字符串转换为十进制数	bitmax	求最大无符号浮点整数
blanks	产生空字符串	char	生成字符串
hex2dec	十六进制字符串转换为十进制	ischar	如果是字符则返回真
strcmp	字符串比较		



32. 结构体函数

函数名	描 述	函数名	描 述
isfield	如果子段属于结构则返回真	struct	生成结构数组
isstruct	如果是结构则返回真		

33. 三角函数

函数名	描 述	函数名	描 述
acos	反余弦	acosd	反余弦,结果以角度表示
acosh	反双曲余弦	acot	反余切
acotd	反余切,结果以角度表示	acoth	反双曲余切
acsc	反余割	acscd	反余割,结果以角度表示
acsch	反双曲余割	asec	反正割
asecd	反正割,结果以角度表示	asech	反双曲正割
asin	反正弦	asinh	反双曲正弦
atan	反正切	atan2	四象限反转正切
atand	反正切,结果以角度表示	atanh	反双曲正切
cos	余弦	cosd	余弦,结果以角度表示
cosh	双曲余弦	cot	余切
cotd	余切,结果以角度表示	coth	双曲余切
csc	余割	cscd	余割,结果以角度表示
csch	双曲余割	hypot	平方和的平方根
sec	正割	secd	正割,结果以角度表示
sech	双曲正割	sin	正弦
sind	正弦,结果以角度表示	sinh	双曲正弦
tan	正切	tand	正切,结果以角度表示
tanh	双曲正切		

34. 视频与图像处理模块集函数

函数名	描 述
estimateFundamentalMatrix	由立体图像相关点估计基本矩阵

参考文献

[1] 刘杰. 基于模型的设计及其嵌入式实现[M]. 北京:北京航空航天大学出版社,2010.

[2] Documentation for MathWorks Products, R2010b[EB/OL].
http://www. mathworks. com/access/helpdesk/help/helpdesk. html

[3] The MathWorks,Inc. Simulink for Simulation and Model-Based Design[EB/OL].
http://www. mathworks. cn/programs/simulink_cd.

[4] The MathWorks, Inc. DO Qualification Kit 1 User's Guide, October,2010.

[5] The MathWorks, Inc. Embedded IDE Link 4 User's Guide For Use with Texas Instruments' Code Composer Studio,2010.

[6] The MathWorks, Inc. Embedded MATLAB Getting Started Guide, October,2010.

[7] The MathWorks, Inc. Embedded MATLAB User's Guide, October,2010.

[8] The MathWorks, Inc. Fixed-Point Toolbox 3 User's Guide, October,2010.

[9] The MathWorks, Inc. MATLAB 7 Desktop Tools and Development Environment, October,2010.

[10] The MathWorks, Inc. MATLAB 7 Programming Fundamentals, October,2010.

[11] The MathWorks, Inc. Real-Time Workshop 7 Target Language Compiler, October,2010.

[12] The MathWorks, Inc. Real-Time Workshop 7 User's Guide, October,2010.

[13] The MathWors. Real-Time Workshop Embedded Coder 5 Developing Embedded Targets, October, 2009.

[14] The MathWorks, Inc. Real-Time Workshop Embedded Coder 5 Reference, October,2010.

[15] The MathWorks, Inc. Real-Time Workshop Embedded Coder 5 User's Guide, October,2010.

[16] The MathWorks, Inc. Control System Toolbox User's Guide,October,2010.

[17] The MathWorks, Inc. Simulink 7 Reference, October,2010.

[18] The MathWorks, Inc. Simulink 7 User's Guide, October,2010.

[19] The MathWorks, Inc. Simulink 7 Writing S-Functions, October,2010.

[20] The MathWorks, Inc. Simulink Design Verifier 1 User's Guide, October,2010.

[21] The MathWorks, Inc. Simulink Verification and Validation 2 User's Guide, October,2010.

[22] The MathWorks, Inc. Stateflow 7 Getting Started Guide, October,2010.

[23] The MathWorks, Inc. Stateflow 7 User's Guide, October,2010.

[24] The MathWorks, Inc. SystemTest 2 User's Guide, October,2010.

[25] The MathWorks, Inc. Embedded IDE Link For Use with Altium TASKING October,2010.

[26] The MathWorks, Inc. Target Support Package For Use with Infineon C166, October,2010.
[27] Yanik, Paul. Migration from Simulation to Verification. EDA Tech Forum, Newton, MA. 2004.
[28] Grantley Hodge, Jian Ye, Walt Stuart. Multi-Target Modeling for Embedded Software Development for Automotive Applications[C]. 美国:2004 SAE World Congress,2004.
[29] General Motors Powertrain. Automatic Code Generation Process[OL].
http://www.mathworks.com/automotive/iac/presentations/michaels.pdf.
[30] 恒润科技. http://www.hirain.com.
[31] 风标电子. http://www.windway.cn/.
[32] The Microchip,Inc. dsPIC33FJ12GP201/202 Data Sheet,2009,
http://ww1.microchip.com/downloads/en/DeviceDoc/70264D.pdf
[33] The Microchip,Inc. dsPIC33FJ16GP304 Data Sheet ,2009.
http://ww1.microchip.com/downloads/en/DeviceDoc/70290F.pdf.
[34] The Philips Semiconductors,Inc. LPC2124 Datasheet, December 2004.
http://www.keil.com/dd/docs/datashts/philips/lpc2114_2124.pdf.
[35] The Philips Semiconductors,Inc. LPC2103 Datasheet,2006.
http://www.keil.com/dd/docs/datashts/philips/lpc2101_2102_2103.pdf.
[36] The Labcenter Electronics,Inc. Intelligent Schematic Input System User Manual Issue 6.0 . 2002.
[37] RTCA, Inc. Software Considerations in Airborne Systems and Equipment Certification. 美国:RTCA Inc,1992.
[38] Conducting Verification & Validation following DO178B guidelines[OL].
http://www.smartworks.us/datasheets/whitepaper3.pdf.
[39] ROBERT HAMMARSTRöM, JOSEF NILSSON. A Comparison of Three Code Generators for Models Created in Simulink[D]. 哥德堡:Chalmers University of Technology. 2006.
http://tagteamcontent.mathworks.com/tt/sl.ashx?z=501ba437&dataid=9784&ft=1. 2007.
[40] Brett Murphy, Amory Wakefield, and Jon Friedman. Best Practices for Verification, Validation, and Test in Model-Based Design[OL].
http://tagteamcontent.mathworks.com/tt/sl.ashx?z=501ba437&dataid=11031&ft=1. 2008.
[41] Tom Erkkinen. Fixed-Point ECU Code Optimization and Verification with Model-Based Design [OL].
http://www.sae.org/technical/papers/2009-01-0269. 2009.
[42] Jerry Krasner. Model-Based Design and Beyond: Solutions for Today's Embedded Systems Requirements[OL]. http://tagteamcontent.mathworks.com/tt/sl.ashx?z=501ba437&dataid=4484&ft=1. 2004.
[43] Andreas Doblander, Dietmar Gösseringer, Bernhard Rinner. AN EVALUATION OF MODEL-BASED SOFTWARE SYNTHESIS FROM SIMULINK MODELS FOR EMBEDDED VIDEO APPLICATIONS [OL]. http://www.iti.tu-graz.ac.at/download/publications/doblander05a.pdf
[44] Alexander Vikström. A Study of Automatic Translation of MATLAB Code to C Code Using Software From the Mathworks[D]. http://epubl.luth.se/1402-1617/2009/033/LTU-EX-09033-SE.pdf. 2009.
[45] Link for Code Composer Studio Development Tools, The MathWorks, Inc, April 2006.
[46] Jim Tung. 基于模型设计对汽车研发的推动[OL].
http://www.qcdz.cn/upfile/info/2008-8/20080805184840384.pdf.
[47] T. Schattkowsky and W. Mueller. Model-Based Specification and Execution of Embedded Real-

Time-Systems, In Proc. Of the Design, Automation and Test in Europe Conference and Exhibition (2004), pp. 1392—1393 Vol. 2.

[48] M. W. Whalen and M. P. E. Heimdahl. On the Requirements of High-Integrity Code Generation. In Proc. of the 4th IEEE International Symposium on High-Assurance Systems Engineering (1999), pp. 217-224.

[49] G. Karsai, J. Sztipanovits, A. Ledeczi and T. Bapty. Model-integrated development of embedded software. Proceedings of the IEEE 91:1(2003) 145-164.

[50] J. Loyall, Jianming Ye R. Shapiro, S. Neema, N. Mahadevan, S. Abdelwahed, M. Koets, and D. Varner. A case study in applying qos adaptation and model-based design to the design-time optimization of signal analyzer applications. IEEE Military Communications Conference (MILCOM), November 2004.

[51] Return on Investment for Independent Verification & Validation. NASA. 2004.

[52] The MathWorks, Inc. Simulink for Simulation and Model-Based Design [OL]. http://www.mathworks.cn/programs/simulink_cd/. 2008.



[General Information]
书名=基于模型的设计 MCU篇
作者=刘杰著
页数=502
出版社=北京市：北京航空航天大学出版社
出版日期=2011.01
SS号=12761155
DX号=000008060406
URL=http://book.szdnet.org.cn/bookDetail.jsp?dxNumber=000008060406&d=DE5447A3F51E868D9D81AE6440B9EE66